

SIEMENS

SIMATIC NET

CMX Programming Interface

SOFTNET S7 UNIX

User Manual

C79000-G8976-C112

Release 1

Wir haben den Inhalt der Druckschrift auf Übereinstimmung mit der beschriebenen Hard- und Software geprüft. Dennoch können Abweichungen nicht ausgeschlossen werden, so daß wir für die vollständige Übereinstimmung keine Gewähr übernehmen. Die Angaben in der Druckschrift werden jedoch regelmäßig überprüft. Notwendige Korrekturen sind in den nachfolgenden Auflagen enthalten. Für Verbesserungsvorschläge sind wir dankbar.

Technische Änderungen vorbehalten.

We have checked the contents of this manual for agreement with the hardware described. Since deviations cannot be precluded entirely, we cannot guarantee full agreement. However, the data in this manual are reviewed regularly and any necessary corrections included in subsequent editions. Suggestions for improvement are welcome.

Technical data subject to change.

Weitergabe sowie Vervielfältigung dieser Unterlage, Verwertung und Mitteilung ihres Inhalts nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz. Alle Rechte vorbehalten, insbesondere für den Fall der Patenterteilung oder GM-Eintragung.

C79000-G8976-C112
Copyright © Siemens AG 1997
All Rights Reserved

The reproduction, transmission or use of this document or its contents is not permitted without express written authority. Offenders will be liable for damages. All rights, including rights created by patent grant or registration of a utility or design, are reserved.

C79000G8976-C112
Copyright © Siemens AG 1997
All Rights Reserved

Note

We would point out that the contents of this product documentation shall not become a part of or modify any prior or existing agreement, commitment or legal relationship. The Purchase Agreement contains the complete and exclusive obligations of Siemens. Any statements contained in this documentation do not create new warranties or restrict the existing warranty.

We would further point out that, for reasons of clarity, these operating instructions cannot deal with every possible problem arising from the use of this device. Should you require further information or if any special problems arise which are not sufficiently dealt with in the operating instructions, please contact your local Siemens representative.

General

This device is electrically operated. In operation, certain parts of this device carry a dangerously high voltage.

WARNING !



Failure to heed warnings may result in serious physical injury and/or material damage.

Only appropriately qualified personnel may operate this equipment or work in its vicinity. Personnel must be thoroughly familiar with all warnings and maintenance measures in accordance with these operating instructions.

Correct and safe operation of this equipment requires proper transport, storage and assembly as well as careful operator control and maintenance.

Personnel qualification requirements

Qualified personnel as referred to in the operating instructions or in the warning notes are defined as persons who are familiar with the installation, assembly, startup and operation of this product and who possess the relevant qualifications for their work, e.g.:

- Training in or authorization for connecting up, grounding or labelling circuits and devices or systems in accordance with current standards in safety technology;
- Training in or authorization for the maintenance and use of suitable safety equipment in accordance with current standards in safety technology;
- First Aid qualification.

Table of contents

1 Preface	Fehler! Textmarke nicht definiert.
1.1 Product Description	1
1.2 Target Group.....	1
1.3 Summary of Content	1
1.4 Sample Programs	1
1.5 Abbreviations	2
2 Creating CMX Applications	3
2.1 Installation.....	3
2.2 Linking CMX applications	3
2.3 Restrictions	3
3 The CMX transport access system.....	4
3.1 Communication between TS applications	4
3.2 The CMX program interfaces - an overview.....	4
3.2.1 CMX functions for communication (ICMX(L))	5
3.2.2 System and user options.....	6
4 TS applications	7
4.1 Names and addresses of TS applications.....	7
4.1.1 The GLOBAL NAME of a TS application	7
4.1.2 Properties of a TS application	8
4.1.3 The properties LOCAL NAME and TRANSPORT ADDRESS	9
4.2 Structure of a TS application	10
4.3 Compiling and linking TS application programs.....	12
4.4 TS applications, processes, connections	12
4.4.1 TS applications and processes.....	12
4.4.2 Connections and processes	13
5 Event processing and error handling.....	15
5.1 Receiving events.....	15
5.2 Error handling.....	16
5.2.1 Error checking functions.....	16
5.2.2 Format of CMX error messages	16
5.2.3 Decoding error messages	17
6 Attaching to/detaching from CMX	18

6.1 Attaching to CMX	18
6.2 Detaching from CMX	18
6.3 Examples of attaching and detaching a process	19
6.3.1 Example of attaching and detaching a process at ICMX(L)	19
7 Managing connections between TS applications	20
7.1 Establishing a connection	20
7.2 Closing down a connection	22
7.3 Example of setting up and closing down a connection with ICMX(L)	23
7.3.1 Examples of establishing a connection with ICMX(L)	23
8 Transmitting data	26
8.1 Sending and receiving normal data	26
8.2 Examples of transmitting normal data	27
8.2.1 Example of transmitting normal data via ICMX(L)	28
8.3 Sending and receiving expedited data	28
8.4 Flow control of normal and expedited data	29
9 8 The ICMX(L) program interface	32
9.1 Overview of the program interface	32
9.2 States of TS applications and permissible state transitions	41
9.2.1 Explanations of the possible state transitions	46
9.3 Restrictions associated with specific types of network connection	47
9.4 System options and message length	49
9.4.1 Programming notes	50
9.5 Conventions	51
9.6 ICMX(L) - function calls	51
9.6.1 t_attach - Attach a process to CMX (attach process)	52
9.6.2 t_callback - Register a callback routine	56
9.6.3 t_concf - Establish connection (connect confirmation)	59
9.6.4 t_conin - Receive connection request (connect indication)	61
9.6.5 t_conrq - Request connection (connection request)	64
9.6.6 t_conrs - Respond to connection request (connection response)	68
9.6.7 t_datago - Release the flow of data (data go)	71
9.6.8 t_datain - Receive data (data indication)	72
9.6.9 t_datarq - Send data (data request)	74
9.6.10 t_datastop - Stop the flow of data (data stop)	76
9.6.11 t_detach - Detach a process from a TS application (detach process)	77
9.6.12 t_disin - Accept disconnection (disconnection indication)	78
9.6.13 t_disrq - Close down connection (disconnection request)	80

9.6.14 t_error - Error diagnosis (error)	82
9.6.15 t_event - Await or query event (event).....	83
9.6.16 t_getaddr - Query TRANSPORT ADDRESS (get address).....	88
9.6.17 t_getfds - get file descriptors in use	91
9.6.18 t_getloc - Query LOCAL NAME (get local name).....	93
9.6.19 t_getname - Query GLOBAL NAME (get name).....	96
9.6.20 t_info - Query information on CMX (information)	98
9.6.21 t_perror - Output CMX error message in decoded form.....	100
9.6.22 t_preason - Decode and output reasons for disconnection	101
9.6.23 t_setopt - Set options in CMX (set options)	102
9.6.24 t_strerror - Decode CMX error message	104
9.6.25 t_strerror - Decode reasons for disconnection.....	105
9.6.26 t_xdatgo - Release the flow of expedited data (expedited data go).....	106
9.6.27 t_xdatin - Receive expedited data (expedited data indication)	107
9.6.28 t_xdatrq - Send expedited data (expedited data request)	109
9.6.29 t_xdatstop - Block the flow of expedited data (expedited data stop)	111
10 Configuring CMX Connections	113
10.1 Transport Name Service TNSX	113
10.2 TNSX Daemon tnsxd.....	113
10.3 TNSX Compiler tnsxcom	113
10.4 TNS Entries.....	114
10.4.1 TNS Configuration for the OSI Transport Protocol.....	114
10.4.2 TNS Configuration for RFC1006 via TCP/IP	117
10.4.3 Restrictions for Names in TNS Entries	120
10.5 Working with the TNSX Compiler	120
10.5.1 Generating TNS Entries	120
10.5.2 Modifying Existing TNS Entries	120
10.5.3 Deleting Existing TNS Entries	121
10.5.4 Outputting the Address Information of all TNS Entries.....	121
11 CMX Library Trace	123
11.1 CMXTRACE: Controlling the Trace.....	123
11.2 cmxl: How the Trace is Processed.....	124
11.3 Example of Activating and Evaluating the CMX Library Trace.....	124
12 Appendix	125
12.1 Complete list of CMX error messages	125
12.2 List of reasons for disconnection	126

1 Preface

1.1 Product Description

The CMX transport access system CMX (Communication Method UNIX) is the basic product for communication software. CMX enables communication between applications on different computer systems. It is responsible jointly with the CCPs (Communication Control Programs) for communication tasks. CMX can be used to generate application programs that can communicate with other applications irrespective of the underlying transport system.

1.2 Target Group

This manual is intended for programmers who develop TS applications for communication (TS application = Transport Service application). These applications consist of application programs implemented in C.

In order to work with CMX, one must be familiar with the operating system and be proficient in using the C programming language and the C development system. Knowledge of the principles and methods of data communications will also prove helpful, especially with regard to the OSI Reference Model as standardized in ISO 7498.

1.3 Summary of Content

The manual is divided into two parts:

Introduction and Training

Part 1, consisting of chapter 2 to 7, is intended to help you to get acquainted with CMX and focusses on helping the first time user to create TS applications. It describes the mapping of a TS application onto the process concept of your system and the allocation of transport connections to processes of the TS application. The structure of a TS application is explained, showing how it can be divided into three communication phases and how the functions of the program interface are used within these phases. In addition, you will learn how you can use the Transport Name Service (TNSX) to request names and addresses from the address directory and pass them onto CMX, and how to obtain diagnostic information from CMX in the case of errors. To explain the individual programming steps, program fragments have been provided as examples.

CMX Function Calls

Part 2 consists of chapters 8. This chapter describes the CMX programming interface. Every function call is explained in detail along with its parameters. The description is arranged in alphabetical order. The description of each function call begins with a summary of all the information you need to use the relevant function.

The programming interface is described independent of the underlying operating system. Operating system specific details are described in the corresponding product information.

1.4 Sample Programs

The corresponding product comes with several CMX sample applications. The CMX sample applications can be viewed in the `/opt/lib/cmx/demo` directory.

1.5 Abbreviations

Please note the use of the following abbreviations throughout this document:

CMX

Communication Method in UNIX: transport access system to RFC 1006 over TCP/IP and ISO transport protocol.

ICMX bzw. ICMX(L)

Interface Communication Method in UNIX: programming interface to the transport access system CMX. L stands for Local Area Network.

2 Creating CMX Applications

2.1 Installation

The installation process is described in the SOFTNET-S7/UNIX product info.

The installation of the package CMX is mandatory.

The package CCP-TP4 has to be installed when running CMX applications over OSI transport protocol is required.

Anyway, the steps after installation as described in the SOFTNET-S7/UNIX product info have to be done.

2.2 Linking CMX applications

The following libraries und compile options have to be used when linking cmx applications:

HPUX: 10.1:	CFLAGS = -Ae +Z -Wl,+s,+b,/usr/lib LIBS = /usr/lib/libcmx.sl /usr/lib/libnds.sl /usr/lib/libxti.sl
SCO 5.0:	LIBS = /usr/lib/libcmx.so /usr/lib/libsocket.so /usr/lib/libnds.so /usr/lib/libxti.so
SOLARIS SPARC 2.5:	LIBS = /usr/lib/libcmx.so /usr/lib/libsocket.so /usr/lib/libnsl.so
SOLARIS X86 2.5:	LIBS = /usr/lib/libcmx.so /usr/lib/libsocket.so /usr/lib/libnsl.so

2.3 Restrictions

The CMX programming interface including the underlaying transport systems is a port from SINIX to the above stated operating systems. However, not all functions of the CMX programming interface have been ported and released.



Not all CMX functions as described in this manual have been ported and released. In the following all CMX functions not available in the CMX version which comes with SOFTNET-S7/UNIX are stated.

- The support of expedited data is **not** available. The following CMX functions and events are not available:
t_xdatrq, t_xdatin, t_xdatgo, t_xdatstop
- This manual describes the functionality to interrupt the CMX function t_event() when waiting for incoming CMX events. This functionality is not released und may **not** be used in CMX applications.

3 The CMX transport access system

3.1 Communication between TS applications

Any application that wishes to exchange data with another application in some other end system requires the services of a transport system. The transport system performs all the necessary tasks to set up the connection and to transport data over the physical media (lines, computers). Applications that use the services of a transport system are called TS applications.

A TS application should be capable of setting up connections and exchanging data using different transport systems. As far as possible, the TS application should be independent of the underlying transport system used. Transport systems may differ in various respects, e.g. the size of the data unit that each transport system can handle, the format of the partner application's transport address to be passed, and the format of the TS application's address in the local system. For this reason, CMX provides TS applications with a uniform interface to the various transport services. This interface is the program interface ICMX(L). It provides TS applications with access to the services of transport systems that conform to the standards laid down in the OSI Reference Model for open systems, and to the services of TCP/IP via the migration protocol RFC1006. CMX is thus a transport access system.

A TS application that uses CMX functions can thus communicate in a uniform way with:

- other TS applications in the same computer (local communication),
- TS applications in other UNIX computers which use the functions of the transport access system CMX,
- TS applications in systems of other vendors, assuming they conform to the standards prescribed in the OSI model or are connected via TCP/IP by means of the migration protocol RFC1006.

For the programmer, the uniform program interface ICMX(L) means that he or she can develop TS applications independent of specific data transmission characteristics, i.e. only the ICMX(L) functions need to be programmed for communication. These functions can be used to:

- attach the TS application to CMX,
- set up transport connections to partner applications,
- send and receive data,
- control the data flow,
- disconnect transport connections,
- detach TS applications from CMX.

TS applications that use the functions of CMX interfaces are also called CMX applications in the description below. This term is always used whenever it is necessary to make a distinction between TS applications running under ICMX and other TS applications.

3.2 The CMX program interfaces - an overview

CMX provides the programmer of TS applications with functions for connection-oriented communication. These functions cover local services, connection handling and data exchange. They are available via the ICMX(L) program interface.

The program interfaces of CMX are library interfaces, i.e. the functions of CMX are provided in the form of modules in a library. These modules are linked into the programs of the TS application. All modules are stored in the library `/usr/lib/libcmx.so` (a shared object).

3.2.1 CMX functions for communication (ICMX(L))

The program interface ICMX(L) includes all functions which are used by a TS application for communication.

The following function groups are provided at the ICMX(L) interface:

Functions for attaching to and detaching from CMX

When a TS application attaches itself to CMX, it passes its LOCAL NAME, i.e. its own address within the local system, to CMX. Only then is the TS application addressable. After communication, the TS application must detach itself from CMX.

Functions to establish a connection

This includes functions for

- active connection setup:

The two functions in this group are used to request a connection with a remote TS application (connection request) and to set up the connection after receipt of a positive response from the remote TS application (connection confirmation).

- passive connection: The two functions in this group serve to accept a connection setup request from a remote TS application (connection indication) and to respond to this request (connection response).

Functions to close down a connection

The two functions in this group are used to actively close down a connection (disconnection request) or to accept a disconnection request (disconnection indication).

Functions for the exchange of data

This group of functions allows you to:

- send (data request) and receive (data indication) normal data.
- send (expedited data request) and receive (expedited data indication) expedited data.
Expedited data refers to small amounts of data that can be transmitted to a communication partner with priority over the main data stream. These functions are optional.

Functions for flow control

If you currently cannot or do not wish to receive any data, you can have the data flow stopped by informing CMX. CMX will then stop signaling incoming data. The communication partner is (usually) notified and will not be permitted to send you any further data until you release the data flow. The data flow can be controlled separately for normal and expedited data (datastop, datago, xdatstop, xdatgo).

Functions to request information

This group includes functions that can be used to:

- await or fetch an event (event).
A typical example of an event is a disconnection request from the communication partner.
- request information on errors (error).
- request information on CMX parameters (information).
- query LOCAL and GLOBAL NAMES, and TRANSPORT ADDRESSES (get local name, get name, get address).

Chapters 4 to 7 explain the use of the functions in programs of a TS application. The function calls are described in detail in section 8.6.

3.2.2 System and user options

The functions of CMX consist of mandatory and optional functions with mandatory and optional parameters.

For communication with partners via CMX, the mandatory functions with the mandatory parameters are always available for all transport systems.

Depending on the type of connection to the network, i.e. depending essentially on the transport system, optional functions are also available, as well as optional parameters for the mandatory functions.

The options are the following:

Option	Optional function	Optional parameter	System option	User option
User data at connection setup	no	yes	yes	yes
User data at disconnection	no	yes	yes	yes
Expedited data	yes	yes	yes	yes
Monitoring of inactive time	no	yes	yes	yes
Connection limit active/passive mode	no	yes	no	yes
User reference of attachment	no	yes	no	yes
User reference of connection	no	yes	no	yes
Time limit on synchronous event processing	no	yes	no	yes

Table 1: CMX options

The system options are oriented to the functionality of the transport system. If options are used that the transport system or the communication interface of the partner application does not provide, the connection will not be established, or a disconnect indication will be issued by CMX. Given an appropriate transport system, CMX guarantees error-free execution of your CMX application.

If communication is to be error-free, the user options must also be correct, i.e. the partners must have a common understanding of how they are used.

This means that CMX does not compensate for the difference between the functionality expected in the TS application and that actually provided by the transport system. This applies particularly to the system options shown above.

4 TS applications

This chapter outlines the characteristics of TS applications that use the functions of the CMX program interfaces.

The following points are covered in the sections of this chapter:

- Name and properties of a TS application

Every TS application has a GLOBAL NAME, with which it can be uniquely identified within the network. To communicate with other TS applications in the network, a TS application must be addressable. For this reason, a TS application is assigned the properties TRANSPORT ADDRESS and LOCAL NAME in addition to other properties.

- Structure of a TS application

A TS application is a C program or a system of C programs that calls CMX functions. This section describes what is required when writing TS application programs, how such C programs are compiled, and which libraries must be linked into the source code.

- Association between a TS application, processes, and connection

This section deals with the question of how a TS application can be mapped onto a system's process concept, and illustrates the association between a process and a connection.

4.1 Names and addresses of TS applications

Every TS application has a GLOBAL NAME. This name identifies the TS application uniquely in the network, i.e. different TS applications have different GLOBAL NAMES. The GLOBAL NAME specifies which TS application is involved.

The GLOBAL NAMES of all TS applications in the local system and those of all TS applications in remote systems with which the local TS applications wish to communicate are recorded in a name and address directory. This directory is known as the TS directory. The properties of a TS application are stored in the TS directory along with its GLOBAL NAME. A property refers to any information on the communication partners that may be required by the respective transport system in order to set up a connection. The transport address of a TS application is a typical example of one of its properties.

Within a TS application, only the GLOBAL NAMES of the two communication partners are used. This makes TS applications independent of the specific addressing requirements of the transport system and of changes within the network. All that is needed is the addition or modification of the relevant properties in the TS directory. The TS application reads the properties from the TS directory with the aid of certain ICMX(L) function calls and passes them on to CMX directly (i.e. unseen).

Properties must be managed and GLOBAL NAMES must be assigned by the TNSX administration. It must ensure that among the GLOBAL NAMES of all TS applications no two are the same, i.e. different TS applications must have unique GLOBAL NAMES.

4.1.1 The GLOBAL NAME of a TS application

The GLOBAL NAME of a TS application is a hierarchically structured name consisting of up to 5 name parts: name part[1] through name part[5]. Of these, name part[1] is the highest in the hierarchy, name part[5] the lowest. All levels of the hierarchy need not be present in a GLOBAL NAME; it is possible to omit name parts. A GLOBAL NAME can also consist of a single name part at any hierarchy level. Apart from the hierarchical order, the TNSX makes no further specifications regarding the meanings of the name parts within the GLOBAL NAME.

A few examples of GLOBAL NAMES are given below:

	Namepart[1]	Namepart[2]	Namepart[3]	Namepart[4]	Namepart[5]
GLOBAL NAME 1	D	Siemens AG	Mch-P	DF1	G._Meier
GLOBAL NAME 2		Abteilung A	Reg18	Proz. 1	\$DIALOG
GLOBAL NAME 3	49	089	636		47658

Table 2: Examples for Global Names

4.1.2 Properties of a TS application

The properties of a TS application constitute all the information on a TS application that is required by the transport system in order to set up the connection and manage the actual transmission of data. Properties are assigned to the GLOBAL NAME in the TS directory. The following tables illustrate which properties can be assigned to a TS application in the local system or to the communication partners (i.e. the remote TS applications).

Some of these properties must be queried from TNSX and passed to CMX when attaching the local TS application to CMX or setting up the connection.

Properties of a local TS application

Property	Meaning of the property
LOCAL NAME	This property is needed to attach the TS application to CMX in the local end system. The LOCAL NAME consists of the addresses of the TS application in the local system for the various transport systems. It is made up of a hexadecimal string with non-printing characters.
USER1 USER2 USER3	User-specific properties. Up to 3 user-specific properties may be assigned to each TS application in a freely-selectable format. These properties can be used to store information that is relevant for your application.

Table 3: Properties of a local application

Properties of a remote TS application

Property	Meaning of the property
TRANSPORT ADDRESS	The value of this property is the communication partner's transport address that is expected by CMX at connection setup. It is made up of a hexadecimal string with non-printing characters.
TRANSPORT SYSTEM	The value of this property is the type of transport system used for communicating with a remote communication partner. When writing a TS application you need not concern yourself with this property. CMX uses it internally.
USER1 USER2 USER3	Up to 3 user-specific properties may be assigned to each TS application in a freely-selectable format. These properties can be used to store information that is relevant for your application.

Table 4: Properties of a remote application

From the structure of the properties described above you can see that within the network you can reach a communication partner via exactly one of the types of network connections (station interconnection STA or computer interconnection via WAN or LAN). From the network, a CMX application can be reached by various communication partners via various types of connection. An application program that is being executed in more than one computer must run under a different GLOBAL NAME in each computer. The program name and the GLOBAL NAME of the TS application must not be confused. A TS application has a GLOBAL NAME that is unique within the network. This name is assigned by the network administrator, as required. Nevertheless, the TS application may functionally consist of the same programs in the various end systems.

The properties TRANSPORT ADDRESS and LOCAL NAME, and their meanings, are described in more detail in the following.

4.1.3 The properties LOCAL NAME and TRANSPORT ADDRESS

Every TS application is assigned a unique Transport Service Access Point (TSAP) when it is attached to CMX. The TSAP is identified by means of the LOCAL NAME that is specified by the TS application when it attaches itself to CMX.

The TS application can access the services of the transport system via the TSAP. Which transport systems, i.e. network connections, can be accessed by the TS application will depend on the T-selectors contained in the LOCAL NAME of the TS application. The LOCAL NAME contains one or more T-selectors. A single T-selector can be valid for multiple network connections, provided these are of the same type.

The TS application can be addressed from the network via the T-selector, since the T-selector is a component of its TRANSPORT ADDRESS for the respective network. The TRANSPORT ADDRESS provides a means of uniquely addressing the TS application in the entire network. The TRANSPORT ADDRESS of a TS application consists of the network address of the end system in which the TS application is located and the T-selector of the TS application for this network connection. The TRANSPORT ADDRESS is thus structured as follows:

TRANSPORT ADDRESS = end system network address + (locally unique) T-selector

The following diagram illustrates the relationship between the GLOBAL NAME, TSAP, and TRANSPORT ADDRESS.

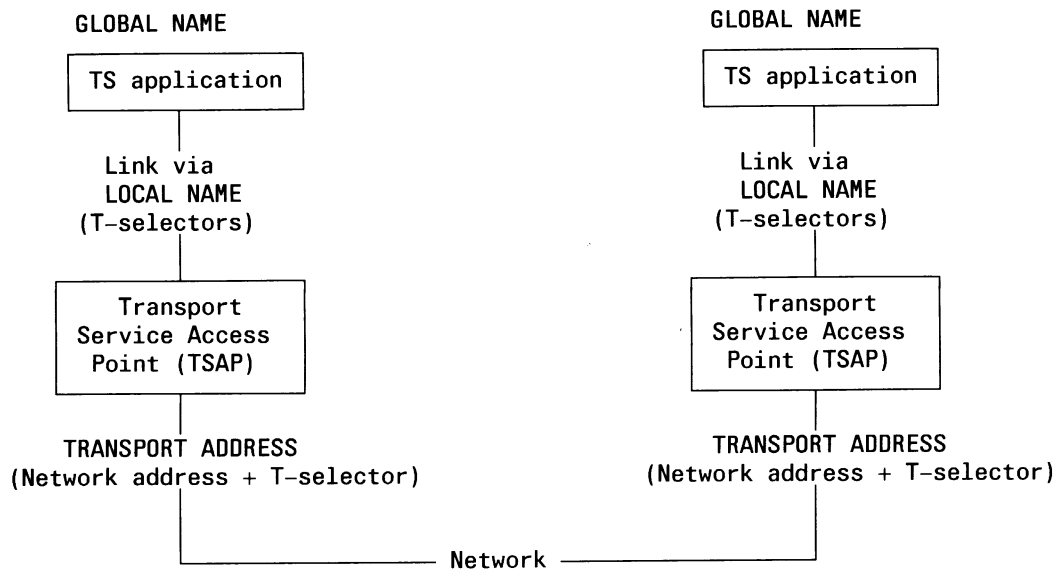


Figure 1: TRANSPORT ADDRESS and LOCAL NAME

The `t_getloc()`, `t_getaddr()` and `t_getname()` calls provided at the ICMX(L) interface can be used to determine the LOCAL NAME or TRANSPORT ADDRESS for a given GLOBAL NAME, and the GLOBAL NAME corresponding to a given TRANSPORT ADDRESS. In other words, all information required in ICMX(L) for a TS application can be queried from the TS directory with the help of these calls.

4.2 Structure of a TS application

A TS application is a C program or a system of C programs that call CMX functions. This chapter describes what should be observed when creating such a program.

The following example illustrates the structure of a program of this type.

```

#include <cmx.h>
#include <tnsx.h>
.
.
main(argc, argv)
int argc;
char *argv[];
{
    .
    .
    /* 1st communication phase */
    t_getloc();          /* Ascertain LOCAL NAME */
    t_attach();          /* Attach to CMX */

    /* 2nd communication phase */
    t_getaddr();         /* Ascertain TRANSPORT ADDRESS */
                        /* of partner */
    t_conrq();           /* Set up connection */
    .
    .
    t_concf();           /* Accept connection */
                        /* confirmation */

    /* 3rd communication phase */
    t_datarq();          /* Send data to partner */
    .
    .
    t_datain();          /* Receive data from partner */
    .
    .
    t_disrq();           /* Close down connection */
    t_detach();          /* Detach from CMX */
    .
    .
    exit();
}

```

Header files

Every TS application program must contain an include statement for the file <cmx.h>. <cmx.h> contains the definitions of the parameters for the functions of the ICMX(L) interface.

All these files are located in the directory /usr/include.

Permissible order for CMX function calls

TS application programs must call CMX communication functions in a certain order. The process of communication can be divided into three phases. A TS application must pass through each phase successfully before it can enter the next phase.

1st communication phase:

The TS application must attach itself to CMX. Only when the TS application is known to CMX can it make use of the services of CMX. The operations of this communication phase are described in chapter 5.

2nd communication phase:

In this phase the TS application sets up the connection to its communication partner. During connection setup the two partners must reach an agreement as to how the subsequent exchange of data is to take place and what form the data is to have. Both partners determine, for example, whether they wish to exchange expedited data. The operations of this communication phase are described in chapter 6.

3rd communication phase:

In the third phase the data is exchanged between the partners. Both communication partners can send and receive data. The operations of this communication phase are described in chapter 7.

This is the order in which a TS application program may call CMX functions. In addition, note that some calls may be issued only after certain responses from the other communication partner have arrived and been received by the TS application (events; see section 4.1).

One might say that a TS application assumes various states during the course of communication. Several states are possible within each communication phase. Only certain transitions are possible between the states within a given phase and between states of different phases.

A TS application can shift from one state to the next only by calling certain CMX functions or when certain events arrive for it from the network.

Sections 8.2 present diagrams showing the possible states and state transitions. These diagrams should make it easier to create TS application programs.

Reaching an agreement as to the form of transferred data

Two TS applications wishing to communicate with each other must also reach an agreement as to the form of the data to be transferred. Any necessary conversion of the data must be performed by the TS applications themselves, since the transfer through the transport systems and CMX is code-transparent.

Parameter passing and storage allocation

In TS applications parameters are passed to CMX functions as values or pointers; for options, unions (union ...) are defined. All structures are declared in the header files.

In your program you must always provide all storage areas used to pass values to CMX or in which CMX is to return anything. You allocate such storage areas either at compile time (statically) or at runtime (dynamically), e.g. with `malloc()` (see the reference manual for the C Development System). In the CMX parameter structures, length fields are defined for areas of variable length. Before calling CMX, enter in these fields the lengths of the areas provided. Then, upon return, you can usually read from these fields the lengths of the data returned by CMX.

4.3 Compiling and linking TS application programs

After a C program `prog.c` for a TS application has been edited, it must be compiled, and the CMX functions from the CMX library `libcmx.so` must be linked. The C Development System is required for this purpose. The C compiler, with included link phase, is called as follows:

```
cc -o prog prog.c ... -lcmx -lsocket -lnsl
```

Please refer to the Release Notice for possible deviations from this syntax.

4.4 TS applications, processes, connections

The two following sections describe the relationships between TS applications and processes and between processes and connections.

4.4.1 TS applications and processes

In the simplest case a TS application is a single process. However, there are additional possibilities for structuring a TS application.

A TS application can work with multiple processes, which need not be related to one another. Each individual process of a TS application must attach to CMX separately. Processes belong to the same TS

application when they have attached themselves to CMX using the same LOCAL NAME. The first process to attach itself creates the TS application.

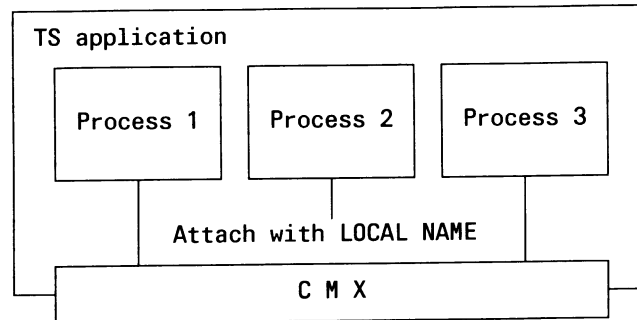


Figure 2: One TS application - multiple processes

On the other hand, one process may control multiple TS applications. To achieve this, you attach the process to CMX using different LOCAL NAMES.

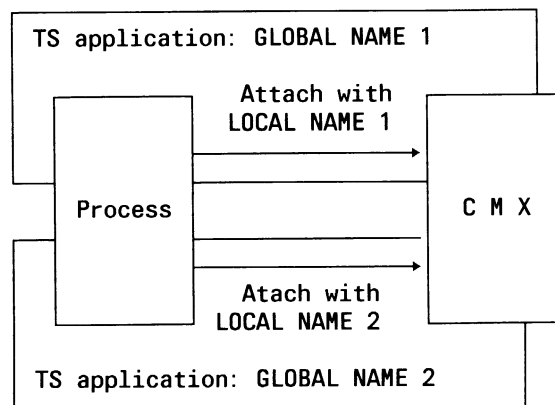


Figure 3: One process - multiple TS applications

The process distinguishes the various TS applications it controls by means of the different LOCAL NAMES or by means of a freely chosen user reference.

4.4.2 Connections and processes

The processes of a TS application can set up connections to other TS applications independently of one another, and individual processes of the TS application may maintain multiple connections simultaneously. If the process is attached to more than one TS application, the connections may also belong to different TS applications. When the connection is set up, a Transport Connection Endpoint (TCEP) is created for each connection. In other words, a single process can serve a number of TCEPs, but the same TCEP may not be simultaneously assigned to multiple processes. Each TCEP is assigned to exactly one process at a given time. It cannot be "inherited" via fork().

CMX assigns each connection an identifier. This is the transport reference. This alone enables the process to address a specific connection.

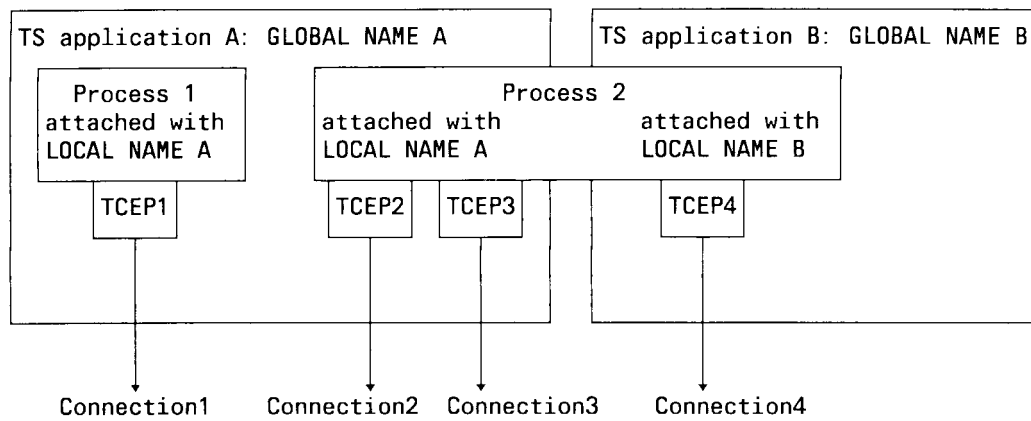


Figure 4: Connections and processes

5 Event processing and error handling

5.1 Receiving events

The operations involved during communications between TS applications are asynchronous, i.e. a wide variety of events can occur independently of the behavior of a TS application. Events are requests and responses received by CMX from other TS applications in the network or messages from the transport systems involved.

Examples of such events are:

- The connection request of a communication partner (the "calling application")
- The arrival of data via an existing connection
- Flow control events (set and released send locks)
- Disconnection by the communication partner or CMX

CMX forwards these events to the TS application when the `t_event()` function is called by the TS application. Exactly one event is passed by CMX for each `t_event()` call, possibly with the identification of the connection involved (transport reference). The TS application must then directly process the received event as required, e.g. by calling the corresponding "fetch" function.

A routine to be called instead of the internal "waiting for/checking events" routine can be passed to CMX using the `t_callback()` call. The program waits in this routine for CMX and program-specific events.

The CMX functions are designed in a manner that allows, but does not compel, the TS application to wait for a possible answer from the network after issuing a call. There are two ways in which a TS application can process events:

Synchronous processing

The TS application calls `t_event()` with the parameter `cmode = T_WAIT`. As long as no event is waiting, the process sleeps and consumes no CPU time. When there is an event (`T_CONIN` in Fig. 7), CMX awakens the process, and `t_event()` returns the code of the event and, when appropriate, the transport reference of the connection involved.

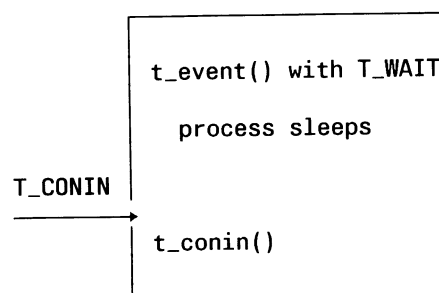


Figure 5: Synchronous processing

When `t_event()` is called it is also possible to limit the waiting time. Simply specify how long the process is to wait for an event. If no event arrives within this time, CMX will resume the process with `T_NOEVENT`.

Asynchronous processing

Call `t_event()` with the parameter `cmode = T_CHECK`. If no event is waiting, the call will immediately return with `T_NOEVENT`. You may continue with any processing and subsequently call `t_event()` again to check for a possible event.

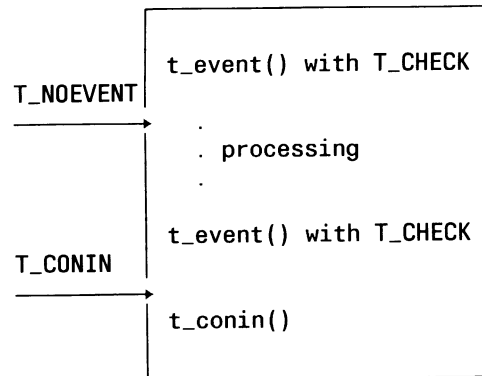


Figure 6: Asynchronous processing

CMX expects a particular reaction, depending on which event was reported. Since program execution is determined by what events occur, the program logic can be largely encapsulated in a switch construction whose cases are the various events (as in the sample programs).

3. Event processing in the program

The `callback()` call can be used to insert your own callback routine. This routine is called instead of the internal "event waiting point" routine during `t_event()`. In the callback routine the program must wait for/check CMX events and can also wait for/check its own events.

5.2 Error handling

5.2.1 Error checking functions

A function call resulting in an error always returns with a global error indicator. A more precise value is obtained by calling the error checking function. The following table shows which function calls and error indicators are applicable to the CMX programming interface:

Interface	Function calls	Global error indicator	Error checking function
ICMX(L)	<code>t_....</code>	<code>T_ERROR</code>	<code>t_error()</code>

The values returned by `t_error()` are in hexadecimal form.

5.2.2 Format of CMX error messages

Every error message at ICMX(L) is passed in the form `0x%x`, where `%x` is an error code with a length of 16 bits. The error code is structured as follows:

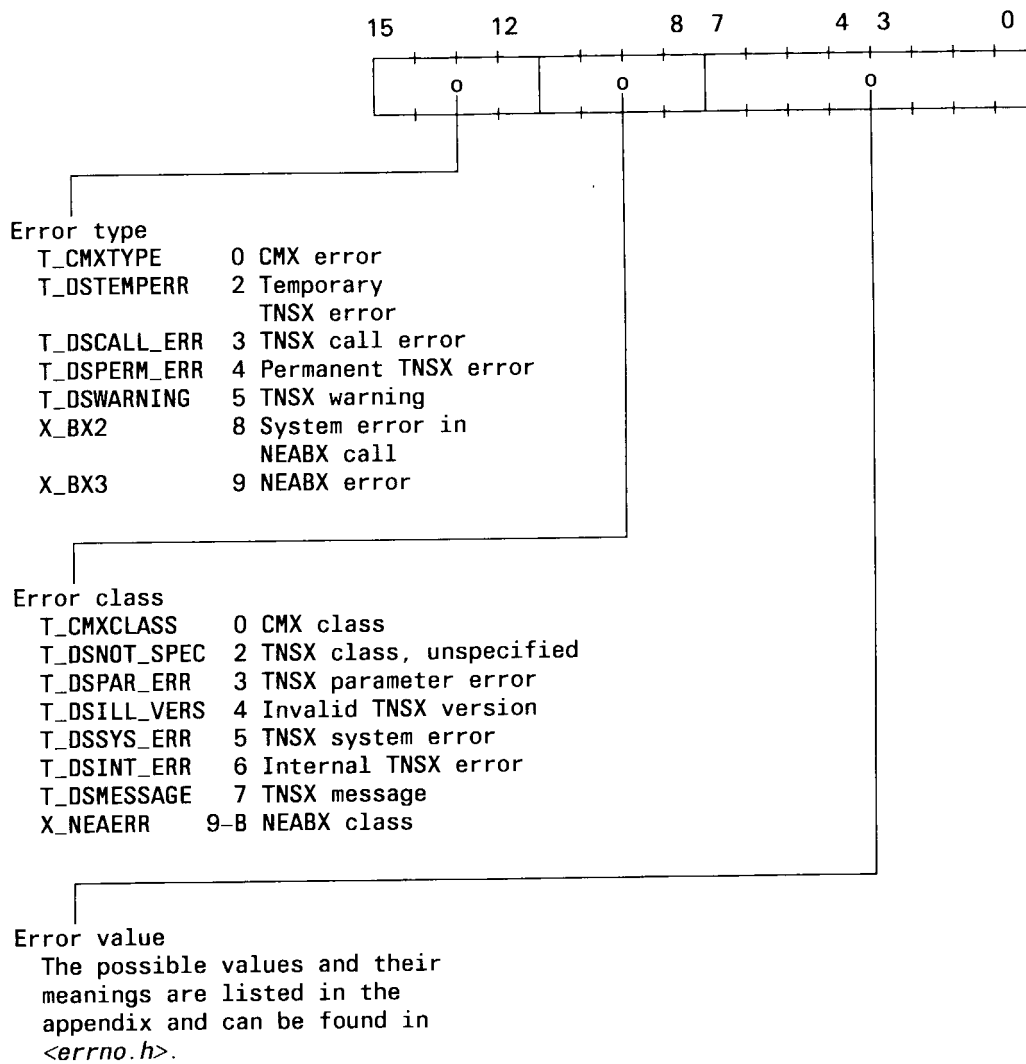


Figure 7: Format of CMX error messages

Error messages are interpreted beginning on the left (bit 15).

The error values for the individual functions of ICMX(L) are listed in the appendix in as far as the errors are generated in CMX. Other error values can be obtained from `<errno.h>`.

5.2.3 Decoding error messages

In addition to the diagnostic functions `t_error()` and `x_error()`, special function calls are provided at the ICMX(L) interface in order to convert error codes into plain English.

The program `cmxdec` can be used at the command level to decode ICMX error messages and the reasons for disconnection.

Reasons for disconnection are the values that are returned in the reason parameter when `t_disin()` is called. They specify why a connection was closed down or rejected.

The error code or the value of reason is decoded by `cmxdec`. The symbolic value defined in the appropriate header file is written to `stderr`. If a corresponding message catalog exists, an explanatory text is also output.

6 Attaching to/detaching from CMX

A TS application comes into existence as soon as a process attaches itself to CMX using the application's LOCAL NAME. Each further process wishing to operate within this TS application must also attach itself to CMX for this TS application, i.e. by using the same LOCAL NAME.

Before a process terminates it must detach itself from CMX. When the last process of a TS application has detached itself from CMX, the TS application no longer exists for CMX.

6.1 Attaching to CMX

A process attaches itself to CMX via the ICMX(L) interface by calling `t_attach()`.

When doing this the process must pass the LOCAL NAME of the TS application for which it wishes to attach itself to CMX.

The process must read the LOCAL NAME from the TS directory prior to attachment, i.e. before the `t_attach()` call. To do this, it calls the ICMX(L) function `t_getloc()` and passes to `t_getloc()` a parameter with the GLOBAL NAME of the TS application for which it wishes to attach itself. `t_getloc()` returns a pointer to a structure in which the LOCAL NAME is stored. This pointer is passed as a parameter in `t_attach()`.

Thus, the `t_getloc()` call must precede the `t_attach()` call.

When the first process of a TS application attaches itself, a Transport Service Access Point (TSAP) is created for the TS application. The TSAP is the point at which the transport service is accessible. It is assigned the LOCAL NAME of the TS application.

When attaching itself, each process of a TS application specifies:

- whether it wishes to actively set up connections for the TS application.
The TS application can then assume the role of the "calling TS application" in the subsequent connection setup phase.
- whether it wishes to wait passively on behalf of the TS application for connection requests from other TS applications in the network. The TS application can then assume the role of the "called TS application" during the course of communication.

The same process can also be attached for several different TS applications. To do this, it must call `t_attach()` and `t_getloc()` for each such TS application.

CMX accepts connection requests from remote TS applications on behalf of a

TS application as soon as a process of the TS application has attached itself to CMX for passive connection setup. Incoming connection requests are initially forwarded by CMX to the process that was the first in the TS application to attach itself for passive connection setup.

Only after successful attachment can a process call other CMX functions, i.e. issue other `t_...()` calls.

6.2 Detaching from CMX

Before a process terminates, it calls `t_detach()`. `t_detach()` detaches the process from CMX for that TS application. First, however, all TS connections maintained by the process must be closed down (see chapter 6). If the process does not do this, CMX implicitly closes down all TS connections itself. This is, however, provided only for exceptional situations, for example when a process is terminated prematurely.

When the last process of a TS application has detached itself, the TS application no longer exists for CMX. Connection requests from remote TS applications will no longer be accepted for that TS application.

6.3 Examples of attaching and detaching a process

6.3.1 Example of attaching and detaching a process at ICMX(L)

The following program fragment shows the program execution sequence when a process is attached and detached at the ICMX(L) interface.

A process attaches itself to CMX for the TS application "Test_application_ACT" and then detaches itself. In the option structure `t_opta1` it specifies that it only wishes to actively set up connections in this TS application (`T_ACTIVE`), and that no more than one connection is to be simultaneously maintained.

```
#include      <stdio.h>
#include      <cmx.h>
#include      <tnsx.h>
.
.
#define ERROR  1
.
.
struct  t_opta1 t_opta1 = { T_OPTA1, T_ACTIVE, 1 };
                                     /* t_attach () */
.
.
/* Structures for addressing */
#define MYNAME "Test_application_ACT"
char myname[TS_LPN+1] = { MYNAME } ;
struct t_myname t_myname, *p_myname;
.
.
/* Attach active application to CMX */
if ((p_myname = t_getloc(myname, NULL)) != NULL)
    t_myname = *p_myname;
else {
    fprintf(stderr, ">>> ERROR 0x%x in t_getloc\n", t_error());
    exit(ERROR);
}
if (t_attach(&t_myname, &t_opta1) == T_ERROR) {
    fprintf(stderr, ">>> ERROR 0x%x in t_attach\n",
            t_error());
    exit(ERROR);
}
fprintf(stderr, "Application '%s' attached.\n", myname);
.
.
/* Detach TS application from CMX */
if (t_detach(&t_myname) == T_ERROR)
    fprintf(stderr, ">>> ERROR 0x%x in t_detach\n",
            t_error());
fprintf(stderr, "Application '%s' detached.\n", myname);
.
.
```

7 Managing connections between TS applications

Connection setup and disconnection involve two TS applications. One is the calling TS application; it initiates connection setup. The other is the called TS application, with whom the calling TS application wishes to establish a connection. The following sections elucidate the relationships and sequences.

The fact that CMX is displayed only once in the diagrams is just a simplification of the presentation. Actually, each partner uses "his" CMX in his processor, and in between stand the network and the transport systems.

7.1 Establishing a connection

The processing sequence in the course of setting up a connection at ICMX(L) is explained first. The following figure illustrates the chronological sequence of ICMX(L) calls in the programs of the calling and called TS application.

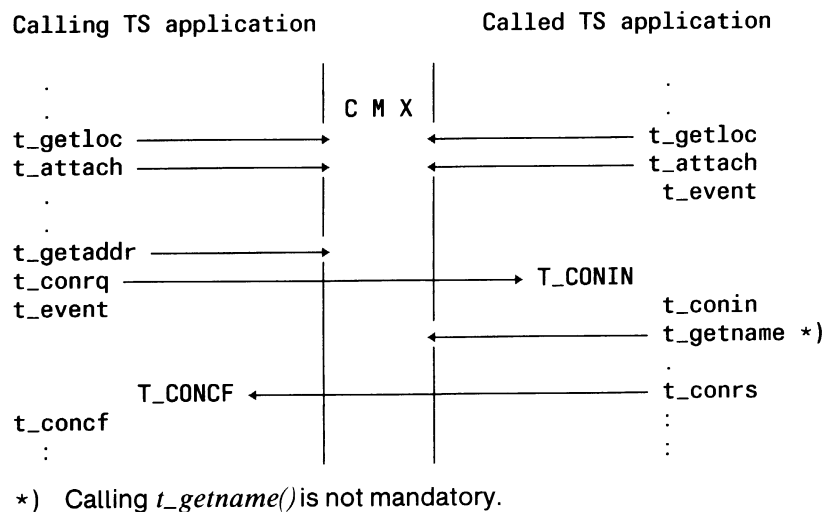


Figure 8: Establishing a connection (ICMX(L))

Course of connection setup in the calling TS application

The process of the calling TS application must inform CMX when attaching itself to it that it intends to actively set up a connection. The calling TS application first obtains its LOCAL NAME and then attaches itself to CMX. Thereafter it ascertains the TRANSPORT ADDRESS of the called TS application and requests a connection using `t_conrq()`.

It then waits with `t_event()` for confirmation of the called TS application, i.e. for the TS event `T_CONCF`. When `t_event()` has reported the TS event, the calling TS application establishes the connection with the call `t_concf()`.

Course of connection setup in the called TS application

Each process of the called TS application must inform CMX when attaching itself to it that it intends to passively set up a connection. After being attached, the called TS application initially waits for a TS event with `t_event()`. The TS event `T_CONIN` indicates the connection request of the calling TS application. The called TS application accepts this connection indication with the call `t_conin()`. It can then ascertain from the TRANSPORT ADDRESS of the calling TS application its GLOBAL NAME (calling `t_getname()`, which is not mandatory), and answers the connection request with `t_conrs()`.

Exchanging user data during connection setup

The reason the calls `t_conin()` (connect indication) and `t_concf()` (connect confirmation) are required is that both TS applications can already exchange user data while the connection is being set up, if the transport system supports this option (see section 2.2.2).

With `t_conrq()` the calling TS application may pass user data, i.e. a small quantity of data that the called TS application receives with `t_conin()`. If the called TS application then answers the connection request with `t_conrs()`, it in turn may also pass information. This is received by the calling TS application with `t_concf()`.

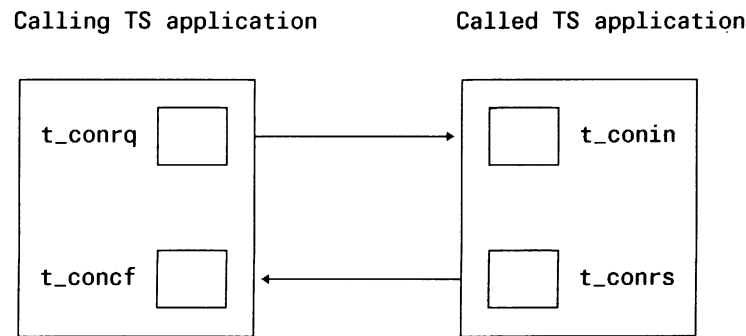


Figure 9: Exchange of user data during connection setup

Rejecting a connection request

The called TS application may also reject the connection request. The sequence is the same. The event `T_CONIN` must first be accepted with `t_conin()`, but instead of the call `t_conrs()` the call `t_disrq()` is issued (see also "Closing Down a Connection").

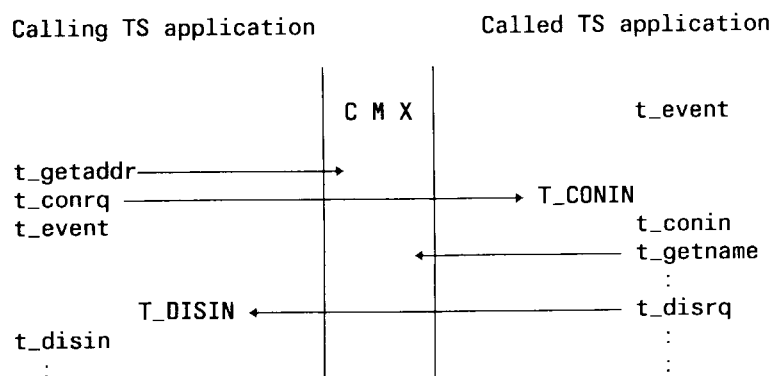


Figure 10: Rejecting a connection request

Agreeing on expedited data

If the transport system provides the expedited data option, the TS applications may agree on its use during connection setup. This takes place as follows:

With the connection request with `t_conrq()` the calling TS application makes a proposal, which the called TS application can only "negotiate down". This means: If the calling TS application proposes not using any expedited data, then this is settled for the connection. If on the other hand it proposes that expedited data be exchanged, the called TS application may accept or reject this in its connection response with `t_conrs()`. In both cases the answer is binding.

If one of the two TS applications does not agree with the result of the expedited data negotiation, it may close down the connection.

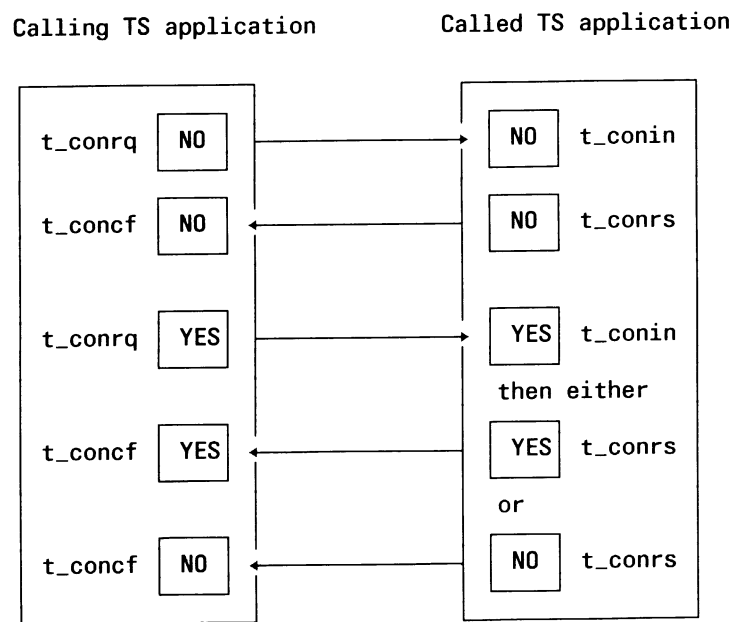


Figure 11: Negotiation regarding expedited data during connection setup

7.2 Closing down a connection

Either of the two communicating TS applications may call `t_disrq()` in order to close down the connection. The partner TS application then receives the event `T_DISIN`.

By calling `t_disin()` it accepts the disconnection. With this call it obtains the reason for the disconnection.

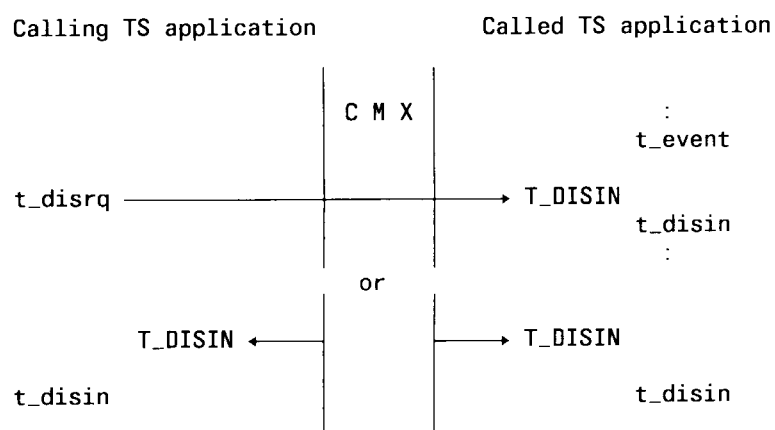


Figure 12: Closing down a connection

If the transport system provides for it, the TS application that closes down the connection may include user data with `t_disrq()`. The partner TS application receives this with `t_disin()`.

The connection may also be closed down by CMX. In this case, both TS applications receive the event T_DISIN, which they must fetch with t_disin(). Based on the reason given for the disconnection each TS application can ascertain whether the connection was closed down by the other TS application or by CMX.

7.3 Example of setting up and closing down a connection with ICMX(L)

7.3.1 Examples of establishing a connection with ICMX(L)

The two following program fragments show how a connection is set up. Example 1 shows the program structure for the calling TS application. Example 2 shows the program structure for the called TS application.

Example 1:

```
#include      <stdio.h>
#include      <cmx.h>
#include      <tnsx.h>
.
.
#define ERROR 1
.
.
int          tref;                /* Transport reference */
int          reason;              /* Reason for disconnection */
/* Structures for addressing */

#define PNAME "Test_applicaton_PAS"
char pname[TS_LPN+1] = { PNAME } ;
struct t_partaddr t_partaddr, p_partaddr;
.
.
/* set up the connection of the passive partner */
if ((p_partaddr = t_getaddr(pname, NULL)) != NULL)
    t_partaddr = *p_partaddr;
else {
    fprintf(stderr, ">>> ERROR 0x%x at t_getaddr\n", t_error());
    exit(ERROR);
}
if (t_conrq(&tref, (union x_address *)&t_partaddr,
            (union x_address *)&t_myname, NULL) == T_ERROR) {
    fprintf(stderr, ">>> ERROR 0x%x at t_conrq, tref 0x%x\n",
            t_error(), tref);
    exit(ERROR);
}
/* event-driven processing */
* t_event() waits synchronously (T_WAIT) */
/*
for (;;) {
    switch (event = t_event(&tref, T_WAIT, NULL)) {
        case T_CONCF:
            /*
             * Connection setup successfull?
             */
            if (t_concf(&tref, NULL) == T_ERROR) {
                fprintf(stderr, ">>> ERROR 0x%x at t_concf
                    tref 0x%x\n",
                    t_error(), tref);
                exit(ERROR);
            }
            fprintf(stderr, "Connection established
```

```

        to '%s' .\n", pname);
        .
        .
    case T_DISIN:
        /* Disconnection by partner or system */
        if (t_disin(&tref, &reason, NULL) == T_ERROR) {
            fprintf(stderr, ">>> ERROR 0x%x at t_disin\n",
                tref 0x%x\n",
                t_error(), tref);
            exit(ERROR);
        }
        fprintf(stderr, "Received disconnect indication,
            tref      0x%x,
            reason %d\n", tref, reason);
        .
        .
    }
}
/* Disconnection */
if (t_disrq(&tref, NULL) == T_ERROR){
    fprintf(stderr, ">>> ERROR 0x%x at t_disrq tref 0x%x\n",
        t_error(), tref);
    exit(ERROR);
}
fprintf(stderr, "Connection tref 0x%x actively closed down.\n",
    tref);
.
.

```

Example 2

The TS application waits passively for an incoming connection request, accepts the connection, and then closes it down.

```

#include      <stdio.h>
#include      <cmx.h>
#include      <tnsx.h>
.
.
#define ERROR 1
.
.
int          tref;                /* Transport reference */
int          reason;              /* Reason for disconnection */
/*
 * Structures for addressing
 */
struct t_myname t_myname, *p_myname;
struct t_partaddr t_partaddr;
.
.
/* Event-driven processing:
 * t_event() waits synchronously (T_WAIT)
 */
for (;;) {
    switch (event = t_event(&tref, T_WAIT, NULL)) {
    case T_CONIN:
        /* Accept connection request */
        if (t_conin(&tref, (union x_address *)&t_myname,
            (union x_address *)&t_partaddr, NULL) ==

```

```

        T_ERROR) {
    fprintf(stderr, ">>> ERROR 0x%x at t_conin
        tref 0x%x\n",
        t_error(), tref);
    exit(ERROR);
}
if (t_conrs(&tref, NULL) == T_ERROR) {
    fprintf(stderr, ">>> ERROR 0x%x at t_conrs tref
        0x%x\n",
        t_error(), tref);
    exit(ERROR);
}
.
.
case T_DISIN:
/*
 * Disconnection by partner or system
 */
if (t_disin(&tref, &reason, NULL) == T_ERROR) {
    fprintf(stderr, ">>> ERROR 0x%x at t_disin tref
        0x%x\n",
        t_error(), tref);
    exit(ERROR);
}
fprintf(stderr, "Received disconnect indication, tref
0x%x,
        reason %d\n", tref, reason);
.
.
}
}
/*
 * Disconnection
 */
if (t_disrq(&tref, NULL) == T_ERROR){
    fprintf(stderr, ">>> ERROR 0x%x at t_disrq tref 0x%x\n",
        t_error(), tref);
    exit (ERROR);
}
fprintf(stderr, "Connection tref 0x%x actively closed down
        .\n", tref);
.
.

```

8 Transmitting data

Once a connection has been set up, the two TS applications can exchange data. Either TS application may initiate the data exchange regardless of whether it is the calling or the called TS application.

The amount of data forming a logical unit from the point of view of the TS applications is referred to as a message, or TSDU (Transport Service Data Unit). A TSDU may be any length (but see also section 8.4).

However, CMX can accept only a limited amount of data at any one time. This is referred to as a data unit or TIDU (Transport Interface Data Unit). The maximum length of a TIDU depends on the transport system. This length must be queried for every connection using the call `t_info()`.

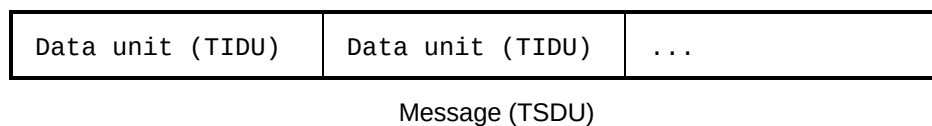


Figure 13: TIDU and TSDU

The logical linkage of TIDUs to form a TSDU is controlled by means of a parameter, which specifies for each TIDU in a message whether it is followed by a further TIDU or is the last one in the TSDU.

If the transport system provides the option, and both TS applications agree to it when the connection is set up, they may also exchange expedited data. Expedited data is a small quantity of data that is given priority over normal data, i.e. expedited data never arrives later than normal data sent subsequently to the expedited data.

Expedited data must always be transmitted all at once. A unit of expedited data is called an ETSDU (Expedited Transport Service Data Unit).

8.1 Sending and receiving normal data

Normal data is sent with the call `t_datarq()`.

In the simplest case data transfer proceeds as follows:

- The sending TS application passes one TIDU to CMX with each call.
- The receiving TS application receives the event `T_DATAIN`. This indicates that data has arrived.
- The receiving TS application must accept the data with the call `t_datain()`.

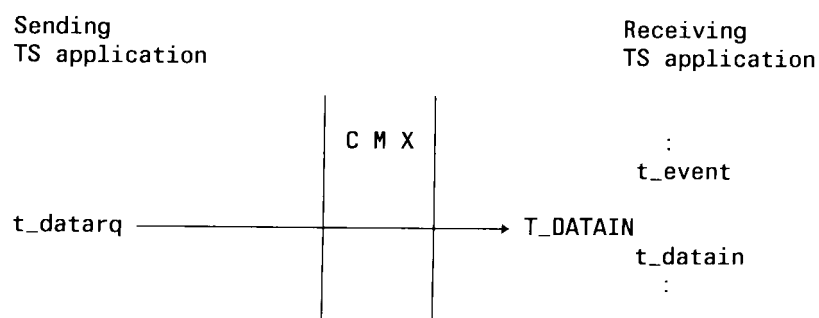


Figure 14: Transmitting normal data

If the TSDU is longer than one TIDU it must be broken down into multiple TIDUs. This is done as follows:

- The sending TS application determines, as sender, when the TSDU is ended. Each time a TIDU is sent with `t_datarq()`, this TS application indicates in the chain parameter whether a further TIDU of the current TSDU is to follow (`chain = T_MORE`) or the TIDU being sent is the last one (`chain = T_END`).
- In the same way, the receiving TS application is informed with each `t_datain()` call by chain as to whether there is another TIDU to come in the current TSDU.

Each TIDU is announced by CMX with a `T_DATAIN` event. However:

There are TIDUs and TIDUs!

The length of a TIDU may be different for each of the two TS applications. Therefore it may happen that the receiving TS application will need to call `t_datain()` less often than the sending TS application calls `t_datarq()` (or vice-versa), because the receiving TS application reads TIDUs in "its" length. The situation then looks like this:

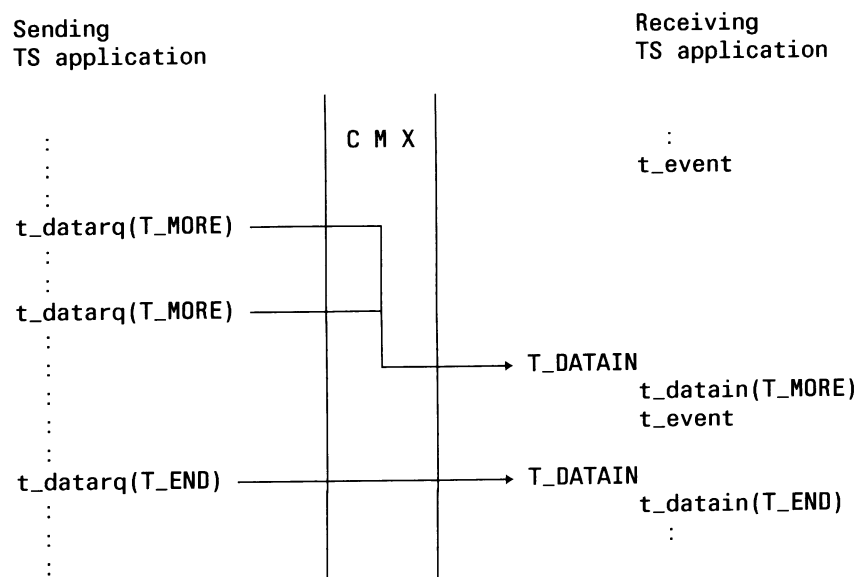


Figure 15: TSDU in multiple TIDUs

The value returned by `t_datain()`

With `t_datain()` you must specify a length for the incoming data to be read. If the length specified is less than the size of the TIDU for the sender, the value returned by `t_datain()` will indicate the excess length of the data in the waiting TIDU.

If a TIDU has not yet been completely read, `t_datain()` must be called repeatedly until the TIDU has been completely read. During this time, `t_event()` may not be called nor may the data flow be controlled.

Note that CMX does not guarantee that at the receiving TS application all TIDUs of a message will be completely filled, even when the size of a TIDU is the same for both the sending and the receiving TS application and the sending TS application sends only completely filled TIDUs.

8.2 Examples of transmitting normal data

The following program fragments illustrate the program execution sequence when transmitting normal data via `ICMX(L)` and `ICMX(NEA)`.

8.2.1 Example of transmitting normal data via ICMX(L)

The TS application receives and sends data. The length of the data is limited here to one TIDU.

```
#include      <stdio.h>
#include      <cmx.h>
#include      <tnsx.h>
.
.
#define ERROR 1
.
.
/* Send and receive buffers */
char    e_bufpt[8000];      /* Receive buffer */
int     e_bufpl;            /* Transfer length */
char    s_bufpt[8000];      /* Send buffer */
int     s_bufpl;            /* Transfer length */
int     chain;              /* TSDU indicator for */
                          /* t_datatq(), t_datain() */
int     tref;               /* Transport reference */
.
.
/* Event-driven processing: */
/* t_event() waits synchronously (T_WAIT) */
for (;;) {
    switch (event = t_event(&tref, T_WAIT, NULL)) {
        .
        .
        /* Receive data; e_bufpl is the TIDU length (t_info()) */
        case T_DATAIN:
            if ((rc = t_datain(&tref, e_bufpt, &e_bufpl, &chain)) ==
                T_ERROR) {
                fprintf(stderr, ">>> ERROR 0x%x in t_datain
                    tref 0x%x\n", t_error(), tref);
                exit (ERROR);
            }
            :
        }
    }
}
/* Send data; s_bufpl is maximum TIDU length */
if ((rc = t_datatq(&tref, s_bufpt, &s_bufpl, &chain)) ==
    T_ERROR) {
    fprintf(stderr, ">>> ERROR 0x%x in t_datatq tref
        0x%x\n", t_error(), tref);
    exit(ERROR);
}
```

8.3 Sending and receiving expedited data

If the exchange of expedited data was agreed at connection setup (see section 6.1), the TS applications may do so as follows:

Expedited data is sent with the call `t_xdatq()`. In the simplest case the sequence is as follows:

- The sending TS application sends expedited data with a call.
- The receiving TS application receives the event `T_XDATIN`. This indicates that expedited data has arrived.
- The receiving TS application must accept the data with the call `t_xdatin()`.

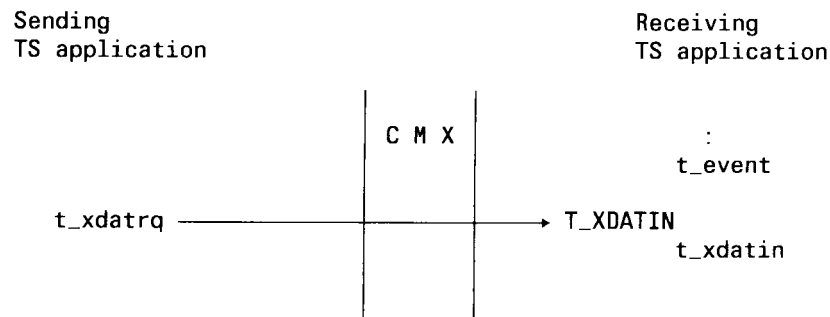


Figure 16: Transmitting expedited data

The value returned by t_xdatin()

With t_xdatin() a length must be specified for the incoming expedited data to be read. If the length specified is less than the amount of expedited data that has arrived, the value returned by t_xdatin() will then give the excess length of the waiting expedited data.

If the expedited data has not yet been completely read, t_xdatin() must be called repeatedly until the data has been completely read. During this time t_event() may not be called nor may the data flow be controlled.

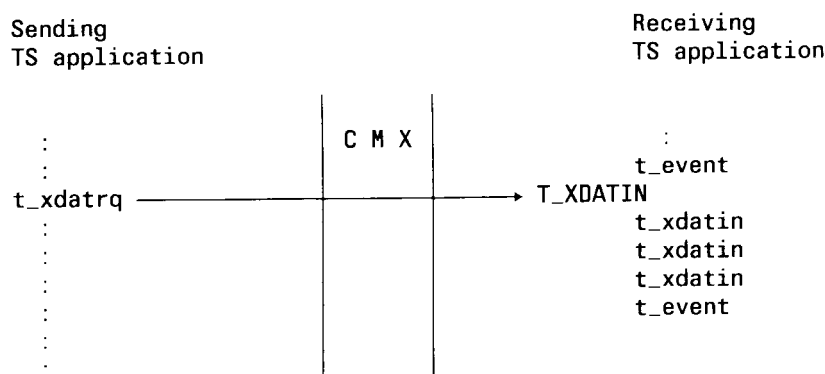


Figure 17: Reading expedited data in piecemeal fashion

8.4 Flow control of normal and expedited data

If a TS application is not ready to receive data over a connection, it informs CMX of this with the call t_datastop(). CMX immediately stops delivering the event T_DATAIN for that connection. For t_datarq() the communication partner will receive the return value T_DATASTOP from CMX and may not send any more data.

As soon as the TS application is again ready to receive data over the connection it calls t_datago(). The communication partner will receive the event T_DATAGO and the TS application may again receive data from it. It again receives the event T_DATAIN.

Flow control for expedited data takes place in the same way. Here the calls t_xdatstop() and t_xdatgo() are used. The corresponding events are T_XDATIN and T_XDATGO.

Note however:

When the flow of expedited data is stopped (with `t_xdatstop()`), CMX also implicitly stops the flow of normal data. When the flow of expedited data is then released again (with `t_xdatgo()`), the flow of normal data remains blocked! It must be expressly released (with `t_datago()`).

When the flow of normal data is released CMX implicitly also releases the flow of expedited data again. Thus, after calling `t_xdatstop()`, calling `t_datago()` releases both the flow of normal data and the flow of expedited data.

What do you gain by preventing `T_DATAIN` or `T_XDATIN` from being received?

During this time the TS application can issue other CMX calls, e.g. to set up a further connection. This would not be possible if a `T_DATAIN` event were waiting. If this were the case, and the TS application did not fetch the data, every `t_event()` call would again return the event `T_DATAIN` and the TS application would not be able to receive the event `T_CONCF`, required to set up a connection.

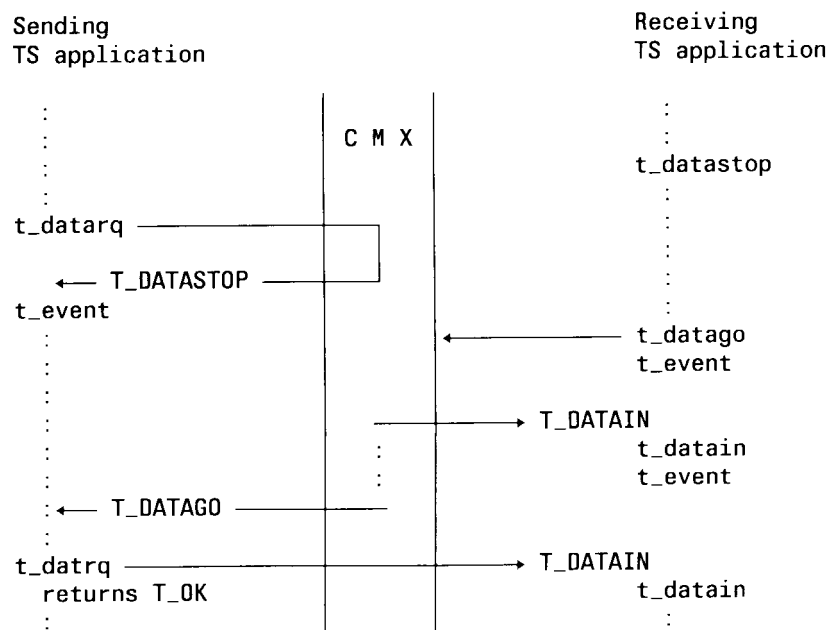


Figure 18: Data flow control at the sending end

The sending TS application receives `T_DATASTOP` in response to the call `t_datarq()`, because the receiving TS application has stopped the data flow or because there is a temporary resource bottleneck in CMX. The data was sent, but no longer indicated to the receiving TS application. The sending TS application must now wait with `t_event()` for the event `T_DATAGO`, in order to be able to send data again.

9 8 The ICMX(L) program interface

This chapter describes the ICMX(L) program interface to the communication method CMX. It contains:

- A summary of the functions of the ICMX(L) interface, with details on the communication phases,
- Notes on the correct use of the functions (finite-state automata),
- A list of the restrictions for station interconnection and computer interconnection via RFC1006 over TCP/IP,
- Notes on the availability of the system options for the transport systems,
- Precise descriptions of the ICMX(L) function calls, with all parameters, in alphabetical order.

9.1 Overview of the program interface

Transport Service ISO 8072

With ICMX(L), the present version of CMX provides a program interface to the connection-oriented transport service (TS) as defined in ISO 8072 within the framework of the OSI Reference Model for open systems. Therefore in ICMX(L) the services T-CONNECT (connection setup), T-DISCONNECT (disconnection), T-DATA (data exchange) and T-EXPEDITED-DATA (exchange of expedited data) are defined with the primitives:

T-CONNECT.request
T-DISCONNECT.request
T-CONNECT.indication
T-DISCONNECT.indication
T-CONNECT.response
T-CONNECT.confirmation
T-DATA.request
T-DATA.indication
T-EXPEDITED-DATA.request
T-EXPEDITED-DATA.indication

Table 5: Service Primitives in ICMX(L)

In addition, ICMX(L) provides local services that simplify the implementation of TS applications. These are:

T-ATTACH

Attach a TS application to CMX

T-DETACH

Detach a TS application from CMX

T-ERROR

Query errors

T-FLOWCONTROL

Flow control for normal data

T-EXPEDITED-FLOWCONTROL

Flow control for expedited data

T-EVENT

TS event check

T-INFO

Information

T-GETADDR

Query TRANSPORT ADDRESS

T-GETLOC

Query LOCAL NAME

T-GETNAME

Query GLOBAL NAME

T-CALLBACK

Register callback routine

T-SETOPT

Set options

The TS permits two TS applications to exchange messages over a transport connection (TC). This connection-oriented communication provides for the exchange of messages without loss or duplication while maintaining the message sequence. Furthermore, by means of connection identification the connection-oriented TS makes it possible to dispense with transferring and processing addresses in the data phase. An established TC is uniquely identified (in both end systems) by a transport reference (tref) between CMX and TS application. Certain parameters that influence message transport on a TC can be negotiated between the TS applications at connection setup. For the correct functioning of communication certain rules must be observed, which are described in the following.

ICMX(L) is implemented as a set of C functions, which make communication between TS applications independent of the specific characteristics of the transport systems used (layers 1 - 4 in the OSI Reference Model) with regard to profile, protocol classes, etc.

Depending on which TS is used, each TC is assigned one or more special files, which are visible to the TS application only in that they consume the corresponding number of the available file descriptors. These special files simplify the cleanup measures taken in CMX following premature termination of the TS application.

Names and addresses

Every TS application has a GLOBAL NAME. This name uniquely identifies the TS application in the network. GLOBAL NAMES are assigned by the TNSX administration. It must ensure that the names of all TS applications are different from one another.

A TS application works exclusively with GLOBAL NAMES. A TS application obtains information from its GLOBAL NAME using CMX calls, e.g. the LOCAL NAME it must specify when attaching to CMX. A TS application can use the GLOBAL NAME of a remote TS application to ascertain the TRANSPORT ADDRESS it must pass to CMX at connection setup.

The LOCAL NAME links the local TS application to a Transport Service Access Point (TSAP). The TRANSPORT ADDRESS of the remote TS application is required to address the Transport Service Access Point (i.e. the TS application linked to it) in the partner system. The LOCAL NAME and TRANSPORT ADDRESS are read from the TS directory.

ICMX(L) functions for querying information from the TS directory are:

`t_getaddr()`

Given the GLOBAL NAME of a TS application, returns its TRANSPORT ADDRESS. The TRANSPORT ADDRESS must be passed through as a parameter to the relevant ICMX(L) call.

`t_getname()`

Given a TRANSPORT ADDRESS, returns the GLOBAL NAME of the TS application.

`t_getloc()`

Given the GLOBAL NAME of a TS application, returns its LOCAL NAME in the current end system. The LOCAL NAME must be forwarded as a parameter to the relevant ICMX(L) call.

<cmx.h> defines the structures `t_myname` and `t_partaddr`. `t_myname` is used by a TS application to receive (pass) its LOCAL NAME from (to) the TNSX; `t_partaddr` is used for the TRANSPORT ADDRESS.

The contents of these structures are as follows:

```

struct t_myname {
    char t_mnmode;           /* = T_MNMODE */
    char t_mnres;            /* = 0 */
    short t_mnln;           /* Length of the filled-in part of
                             the t_myname structure */
    char t_mn[T_MNSIZE];     /* Field for the T-selectors of the
                             LOCAL NAME */
}

struct t_partaddr {
    char t_pamode;           /* = T_PAMODE */
    char t_pares;           /* = 0 */
    short t_palng;          /* Length of the filled-in part of
                             the */
                             /* t_partaddr structure */
    char t_pa[T_PASIZE];     /* Field for the partner address */
}

```

The meanings of members in the structure `t_myname` are shown below:

`t_mnmode = T_MNMODE`

specifies that the field `t_mn` contains a LOCAL NAME.

`t_mnres, t_mn[T_MNSIZE]`

are of no relevance to you. The contents of these members are simply taken from the TNSX and passed on to CMX.

`t_mnln`

specifies the length of all data passed in the structure `t_myname`.

The meanings of members in the structure `t_partaddr` are as follows:

`t_pamode = T_PAMODE`

specifies that the field `t_pa` contains a TRANSPORT ADDRESS.

`t_pares, t_pa[T_PASIZE]`

are of no relevance to you. The contents of these members are simply taken from the TNSX and passed on to CMX.

`t_palng`

specifies the length of all data passed in the structure `t_partaddr`.

The LOCAL NAME and TRANSPORT ADDRESS are passed to CMX or received from CMX in the union `t_address`.

```

union t_address {
    struct t_myname tmyname;
    struct t_partaddr tpartaddr;
}

```

Error handling and diagnosis

All function calls return a return code. This is either T_OK, to indicate successful completion, or T_ERROR to generally indicate that an error occurred. The error check function t_error(), called immediately following an error, returns more detailed diagnostic information. All errors detected by CMX as violations of the communications rules by the TS application have specific error codes and are defined in <cmx.h>. Other errors result from failures in calling functions in the operating system environment in CMX; they can be identified from <errno.h>. The transport systems used generate no error messages; any errors result in disconnection with a corresponding reason. The reason for disconnection is obtained by the TS application when t_disin() is called.

The following functions return the text version of an error code returned by t_error():

t_strerror()

Returns a pointer to the text string for an error code received from ICMX(L).

t_perror()

Calls t_strerror() to ascertain the text string for an error code received from ICMX(L) and writes the string to stderr.

The following functions return the text for a disconnection reason returned by t_disin():

t_strreason()

Returns a pointer to the text string for a disconnection reason that has been received. The reason for disconnection is passed to the TS application when t_disin() is called.

t_preason()

Calls t_strreason() to ascertain the text string for a disconnection reason that has been received with disin() and writes the string to stderr.

The error decoding program cmxdec provides mechanisms for obtaining these texts at the command line.

For diagnostic purposes ICMX(L) provides a trace facility. It can be flexibly controlled via the environment variable CMXTRACE. The trace mechanism logs the calls with their arguments in compressed form in temporary files. The editing program cmxl then converts the log to readable form in a separate step.

TS applications, transport connections and processes

A TS application is a system of programs that uses the TS, i.e. the services of CMX. The mapping of a TS application to the process concept of the system is left up to the implementor. A TS application may organize itself into one or more (not necessarily related) processes. The processes may, essentially independently from one another, maintain TCs to remote TS applications.

One process may also simultaneously control multiple TS applications. In this case, the implementation must provide for suitable coordination of the execution of the various TS applications. CMX supports this through its asynchronous processing mode.

Synchronicity and asynchronicity; TS events

Communications operations are by nature asynchronous: A wide variety of TS events can occur independently of the activity of a TS application. For example, a TS application may be sending data over one TC when, asynchronously, a disconnection indication for another TC arrives, of which the TS application must be informed immediately.

In principle, the functions of CMX are asynchronous: This means, after issuing a call a TS application need not wait for a possible answer from the network. Any answer will be accepted by CMX when it arrives and sent to the TS application as a TS event at the next opportunity when requested.

For this, CMX provides the TS application with a query mechanism in two forms: Synchronous (waiting) and asynchronous (checking). This query mechanism must be appropriately used by the TS application if it wishes to react quickly and properly to TS events.

With synchronous execution, the calling process is suspended until a TS event arrives. This wakes up the process, so that it can immediately process the TS event. Waiting can be limited by specifying a waiting period. The synchronous mechanism is useful for TS applications that maintain several TCs at a time, so that they need not poll them.

With asynchronous execution, at convenient times, such as at the end of a processing step, the process can check whether a TS event has arrived, and handle it before continuing with the next processing step. This is useful for processes that expect longer delays between TS events, during which times they can or must attend to other operations.

The corresponding function in CMX is

`t_event()`

If the parameter value `T_WAIT` is passed, `t_event()` suspends the process until a TS event arrives, the time limit expires or a signal arrives. If a TS event is already waiting, or there is an error, the function returns immediately with the code for the event, or `T_ERROR`. When the time limit expires the process resumes with the TS event `T_NOEVENT`. With the parameter value `T_CHECK`, `t_event()` always returns immediately and returns either the code of the TS event encountered or `T_NOEVENT` or `T_ERROR`.

The following asynchronous TS events are defined in CMX:

`T_NOEVENT`

In the asynchronous case: No TS event present
In the synchronous case: Abort by signal or waiting time elapsed

`T_CONIN`

Arrival of a connection indication from a calling TS application

`T_CONCF`

Arrival of a connection confirmation from a called TS application

`T_DISIN`

Arrival of a disconnect indication from a remote TS application or from CMX

`T_DATAIN`

Arrival of normal data from a remote TS application

`T_XDATIN`

Arrival of expedited data from a remote TS application

`T_DATAGO`

Removal of a block on the sending of normal data and expedited data set through flow control

`T_XDATGO`

Removal of a block on the sending of expedited data set through flow control

`T_ERROR`

Fatal error; more detailed information is provided by the query function `t_error()`.

With each TS event, except for `T_NOEVENT` and `T_ERROR`, the TS application is also given the transport reference, so that it can react for that TC specifically to the TS event.

Some TS events must be accepted by the TS application by calling corresponding functions. Exceptions are: `T_ERROR`, `T_DATAGO`, `T_XDATGO`. Such function calls return additional information on the TS events. The following table lists the TS events and the corresponding functions.

TS event	Function for fetching
----------	-----------------------

T_CONCF	t_concf()
T_CONIN	t_conin()
T_DATAGO	t_event()
T_DATAIN	t_datain()
T_DISIN	t_disin()
T_REDIN	t_redin()
T_XDATGO	t_event()
T_XDATIN	t_xdatin()

Table 6: TS events and functions

As a rule, TS events are delivered in the order in which they occur. Of course, the TS event T_XDATIN may overtake the TS event T_DATAIN, and T_DISIN may overtake T_DATAIN and T_XDATIN. In the latter case the overtaken TS events on that TC are dropped.

Attaching/detaching

Communication by a process via CMX is activated when the process attaches itself to CMX. A special file is opened for the process the first time this is done. This special file is used for exchanging jobs between the CMX library functions and the operating system. A TS application is generated when the first process attaches itself for that TS application. When this is done, a Transport Service Access Point (TSAP) is created, at which the TS is accessible. When the first process is attached the TS application is linked to this TSAP. The TSAP is assigned the LOCAL NAME of the TS application. It thereby becomes addressable from the network. When the TS application is detached, any TCs still in existence are closed down, along with the TSAP; the process environment is dissolved and assigned resources are released for future use.

One and the same process may attach itself for several TS applications at once (i.e. manage multiple TSAPs) and in each of these TS applications maintain multiple Transport Connection Endpoints (TCEP). Also, several processes may attach themselves for the same TS application (use the same TSAP) and actively set up TCs or passively wait for connection indications without interfering with one another. Of course, each TCEP is assigned to exactly one process.

The following functions are used for attaching and detaching. They perform primarily local tasks. If no implicit disconnection must be performed, no information is passed to the network.

t_attach()

Attaches (the current process of) a TS application to CMX. When attached, the process may specify its future behavior in the TS application. The first time a process is attached CMX begins accepting connection indications for the TS application.

t_detach()

Detaches (the current process of) a TS application from CMX. Any existing TCs of the process in the TS application are closed down by CMX. If no more processes of the TS application are attached, the TS application is thereafter no longer known to CMX.

Connection setup and disconnection

Before two TS applications can exchange data, a TC must be set up between them. One of the two TS applications is viewed as the calling TS application; it initiates connection setup. The other is the called TS application; it waits for requests from calling TS applications.

The calling TS application issues a connection request and receives an answer from the called TS application. The called TS application waits for a connection indication (indication of a connection request)

and accepts it or rejects it. During connection setup, the TS applications negotiate certain attributes of the TC for the data transmission and may exchange user data.

The TC may be closed down at any time by either of the TS applications or by CMX. This is not negotiated between the TS applications, but instead is immediately carried out by CMX. The other TS application (or both, if CMX closes down the TC) receives a disconnect indication, which may be neither answered nor averted. CMX indicates all errors in the transport systems by closing down the TCs involved. CMX does not guarantee that data still in transit at the time of the disconnection request will be delivered.

The relevant functions are:

`t_conrq()`

Requests connection setup to the called TS application with the specified TRANSPORT ADDRESS. Reference to the TSAP is established via the LOCAL NAME used when the calling TS application was attached. The function returns immediately after issuing the request; the calling TS application receives a transport reference. It must then wait synchronously or asynchronously for the answer of the called TS application (see above).

`t_concf()`

Accepts from CMX the answer of the called TS application, indicated with T_CONCF; connection setup is now complete.

`t_conin()`

Receives from CMX a connection request, indicated with T_CONIN, from the calling TS application, along with that TS application's TRANSPORT ADDRESS. Reference to the TSAP is established for the called TS application through provision of the LOCAL NAME specified when it was attached.

`t_conrs()`

Answers (accepts) a connection request after it has been indicated with T_CONIN and received by the TS application.

`t_disrq()`

Requests that a connection be closed down; this function may be called at any time by either of the TS applications; it is also used to reject a connection request (instead of accepting it) after the request has been indicated by CMX and received by the TS application.

`t_disin()`

Accepts from CMX the disconnect indication indicated with T_DISIN. The reason for disconnection is also passed to the TS application with this function call.

Data exchange and flow control

Once a connection has been set up, the initiative rests with the TS application (not with CMX). It may:

- send normal data and (if agreed) expedited data, or
- indicate, with `t_event()`, that it is ready to receive normal data or (if agreed) expedited data.

Data transfer is message-oriented: The TS applications exchange Transport Service Data Units (TSDU) - messages of any length - or Expedited Transport Service Data Units (ETSDU) - expedited data of limited length. Expedited data is limited to a few bytes; when transferred it is given priority over the stream of normal data and placed into separate queues. CMX guarantees only that expedited data will never arrive at the receiving TS application later than normal data sent subsequently. At most one complete ETSDU may be passed to CMX per call.

A TSDU (which in principle may be any length) is passed to CMX in portions the length of one Transport Interface Data Unit (TIDU). The length of a TIDU is TC-specific and must therefore be queried by CMX for each TC (`t_info()`). Thus, a TSDU may have to be transferred using multiple send calls. A parameter in each send call indicates whether a further TIDU for that TSDU follows (T_MORE) or not (T_END). It cannot be determined from this how a TIDU is packed for transfer or delivery to the receiving TS application. CMX guarantees only that sequential joining of the TIDUs on the receiving side will reproduce the TSDU from the sending side. The TIDU length may be different for the two TS applications and depends on the TC. CMX

does not guarantee that at the receiving TS application any except the last TIDU of a TSDU will be delivered completely filled.

The arrival of a TIDU of a TSDU (or the arrival of an ETSDU) is indicated to the receiving TS application by means of the TS event T_DATAIN (T_XDATIN). The TS application then fetches the TIDU (ETSDU) with a corresponding function call, either completely or in piecemeal fashion. If necessary it may or must issue several similar calls in order to take in one TIDU (ETSDU) from CMX.

The transfer of TIDUs (ETSDUs) is subject to flow control mechanisms, which can be controlled by CMX and the TS applications. The return code T_DATASTOP (T_XDATSTOP) returned when data is sent indicates to the sending TS application that the TIDU (ETSDU) was processed, but the flow of TIDUs (ETSDUs) has been blocked. No further TIDUs (ETSDUs) may be sent until the flow is released again. Release is indicated by means of the TS event T_DATAGO (T_XDATGO).

The receiving TS application stops and starts the flow of TIDUs (ETSDUs) by means of function calls to CMX, which affect the sending TS application as described above.

The following functions implement data exchange and (active) flow control:

t_datarq()

Requests transfer of a TIDU (possibly partially filled) from a contiguous storage area. The return code T_DATASTOP signifies that the flow is blocked; further send requests are rejected with an error until the flow is released again.

t_datain()

Accepts the data of a TIDU from CMX, placing it into a contiguous storage area, after the TIDU has been indicated with T_DATAIN. The return code specifies how much data is still contained in the current TIDU, so that a TIDU can be read in piecemeal fashion.

t_xdatrq()

Requests transfer of an ETSDU (possibly partially filled); the return code T_XDATSTOP signifies that the flow is blocked; further send requests are then rejected with an error, until the flow is released again.

t_xdatin()

Accepts the data of an ETSDU from CMX, after it has been indicated with T_XDATIN. The return code specifies how much data is still contained in the current ETSDU, so that an ETSDU can be read in piecemeal fashion.

t_datastop()

Blocks, from the receiving side, the flow of normal data over a connection; the TS event T_DATAIN will no longer be indicated for this connection by CMX.

t_datago()

Releases, on the receiving side, the (blocked) flow of normal data and expedited data over a connection; the TS events T_DATAIN and T_XDATIN can again be indicated for the connection by CMX.

t_xdatstop()

Blocks, on the receiving side, the flow of expedited data and normal data over a connection; CMX will no longer indicate the TS events T_XDATIN and T_DATAIN for this connection.

t_xdatgo()

Releases, on the receiving side, the (blocked) flow of expedited data over a connection; the event T_XDATIN can again be indicated by CMX for the connection.

Information service

The information service is a local service with which the TS application can query configuration-dependent parameter values from CMX. The information service is implemented with the following function:

`t_info()`

Returns the length of a TIDU for an established TC. The TIDU is normally only established when connection setup is completed.

Central waiting point

TS applications often expect application-specific events in addition to CMX events. In the callback routine, it is possible to wait for CMX events and application events simultaneously. A central waiting point be defined for this purpose.

`t_callback()`

passes a pointer to CMX indicating a routine in the application, which is called during the execution of the `t_event()` call.

Management options

Options can currently only be set in the CMX library.

`t_setopt()`

sets or cancels the trace options of the appropriate application.

9.2 States of TS applications and permissible state transitions

The sequences of operations at the ICMX(L) program interface are represented in the following diagram by means of finite-state automata. The diagram shows the defined states that a TS application may assume during the course of communication and the permissible transitions between these states. With the aid of the diagram it is possible to identify permissible sequences of CMX calls. The diagram shows when and how the processes of a TS application must react to certain events.

In the diagram each state is represented by a rectangle with a double border. The rectangle contains the name of the state.

The surrounding (outer) rectangles represent the three communication phases.

1st communication phase: Attach process

The process exists, but is not yet or no longer attached to CMX.

2nd communication phase: Connection setup

The process is attached to CMX, but no connection exists. A connection can now be set up.

3rd communication phase: Data transfer

The connection has been set up. The process can now send and receive data.

The 3rd communication phase is subdivided by dotted lines into four subareas. These subareas are:

- Send normal data
- Receive normal data
- Send expedited data
- Receive expedited data

When it reaches this phase, at any given time the process is in exactly one state in each subarea. Only certain combinations of states in these subareas are permitted, i.e. a state transition within one subarea may cause a state transition in another subarea. The connections between the individual states in the various subareas can be seen by examining the conditions for state transitions (see below). If the exchange of expedited data has not been agreed for the connection, the process can only assume states of the upper two subareas.

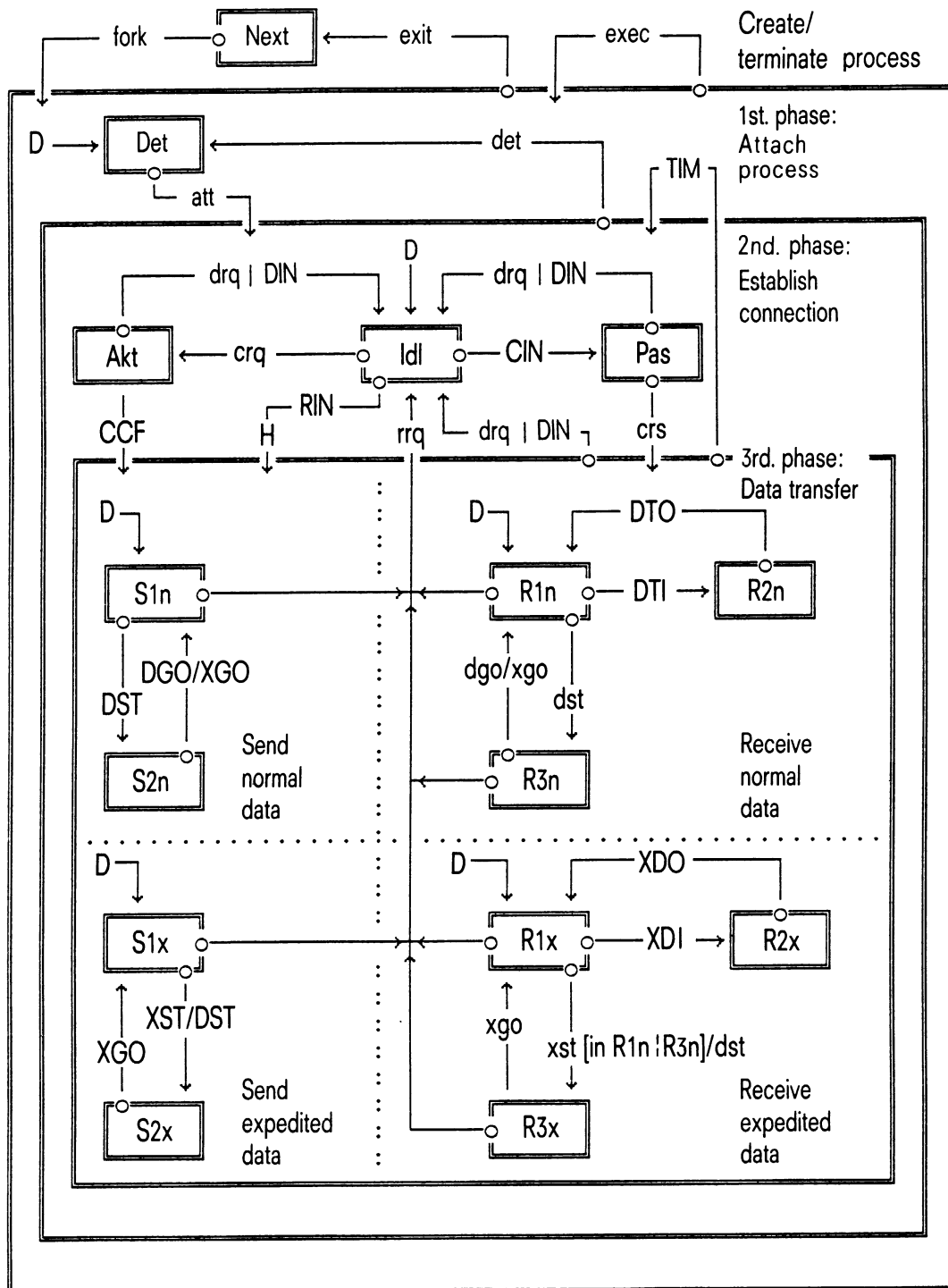


Figure 19: States of TS applications and permissible state transitions

The arrows o-C-> between the rectangles indicate the possible state transitions. C indicates the condition for making the transition from an initial state to the subsequent state (initial state o-C-> subsequent state). Transitions are possible only in the directions indicated by the arrows.

To begin with, the abbreviations used in the diagram are explained below:

Abbreviations for the states:

Nex

The process does not exist (no longer exists).

Det

The TS application is not yet attached to CMX, or the TS application has been detached from CMX.

Idl

Initial state for connection setup and for accepting a connection redirection, or a previously existing connection was closed down.

Act

Waiting for the event T_CONCF following a t_conrq() call (active connection setup).

Pas

A T_CONIN event has arrived (passive connection setup)

S1n

Initial state for t_datarq()

S2n

Normal data flow blocked

R1n

Initial state for t_datain()

R2n

T_DATAIN indicated

R3n

T_DATAIN blocked

S1x

Initial state for t_xdatrq()

S2x

Flow of expedited data blocked

R1x

Initial state for t_xdatin()

R2x

T_XDATIN indicated

R3x

T_XDATIN blocked

Abbreviations for the state transition conditions

fork	Process created
exec	Process shift
exit	Process termination

The state transitions below occur when a CMX function is called:

att	t_attach()
det	t_detach()
crq	t_conrq()
crs	t_conrs()
drq	t_disrq()
dst	t_datastop()
dgo	t_datago()
xst	t_xdatstop()
xgo	t_xdatgo()

The state transitions below occur when an event is accepted:

NET	T_NOEVENT
CIN	T_CONIN
CCF	T_CONCF
DIN	T_DISIN
RIN	T_REDIN

DTI

T_DATAIN

XDI

T_XDATIN

DGO

T_DATAGO

XGO

T_XDATGO

The following state transitions occur when certain return values are returned by CMX functions:

DST

T_DATASTOP returned by t_datarq()

XST

T_XDATSTOP returned by t_xdatrq()

DTO

0 returned by t_datain() (current TIDU completely read)

XDO

0 returned by t_xdatin() (ETSDU completely read)

TIM

t_timeout (inactivity time limit for the connection reached)

9.2.1 Explanations of the possible state transitions

Arrows that terminate at a surrounding rectangle indicate that normally the process first switches to the states indicated by D->.

For example, in the transition to the 3rd communication phase (data transfer) the process initially switches to the states S1n, S1x, R1n, R1x.

Arrows that begin at a surrounding rectangle indicate that a transition is possible from any given state within the rectangle.

State transitions of this kind are:

- fork

If fork() is called in any state of the process, the child process assumes the state Det (process not yet attached to CMX). The state of the parent process remains unaffected.

- exec

If exec() is called in any state of the process, the process switches to the state Det (process detached). It loses all attachments and connections.

- exit

If exit() is called, the process is terminated. All connections are closed down by CMX.

- det

If the process calls `t_detach()` in any state, it switches to the state `Det`. CMX closes down its connections.

- `drq|DIN` (`drq` or `DIN`)

If the process calls `t_disrq()` in any state during data transfer (3rd phase) or during connection setup (2nd phase), the process switches to the state `Idl`. The same thing happens when CMX indicates the event `T_DISIN` to the process. The existing connection is closed down or the connection request of another TS application is rejected.

- `TIM`

If during data transfer the inactivity time limit for the connection, specified by the parameter `t_timeout`, is exceeded, the process switches to the state `Idl` in the 2nd phase.

State transitions within the 3rd phase (data transfer)

The following describes the connections between state transitions in the subareas of the 3rd phase. The state assumed by a process in the subarea "Send normal data" depends on its state in the subarea "Send expedited data", and vice-versa. The state assumed by a process in the subarea "Receive normal data" depends on its state in the subarea "Receive expedited data", and vice-versa.

The following connections exist between the states of the four subareas:

DGO/XGO (DGO initiates XGO)

The event `T_DATAGO` initiates `T_XDATGO`. Along with normal data flow of expedited data is released, assuming it was blocked. Thus, the state transition `S2n -> S1n` initiates the state transition `S2x -> S1x`.

XST/DST (XST initiates DST)

The event `T_XDATSTOP` initiates the event `T_DATASTOP`. The state transition `S1x -> S2x` brings about the state transition `S1n -> S2n`. Blocking the expedited data flow causes blocking of normal data flow.

dgo/xgo (dgo initiates xgo)

If the process calls `t_datago()` in the state `R3n` (`T_DATAIN` blocked), `t_xdatgo()` is implicitly called. The state transition `R3n -> R1n` initiates the state transition `R3x -> R1x`, if the process had previously assumed the state `R3x`.

xst[in R1n|R3n]/dst

If the process is in the state `R1x`, it may call `t_xdatstop()` only if it is in the state `R1n` or `R3n` in the subarea "Receive normal data". It thereby initiates `t_datastop()`. This means the flow of expedited data can be blocked by the process only so long as no `T_DATAIN` is indicated. Along with the flow of expedited data the flow of normal data is implicitly blocked (`R1x -> R3x` initiates `R1n -> R3n`).

9.3 Restrictions associated with specific types of network connection

This section describes restrictions associated with the use of station interconnection of UNIX computers and computer interconnection via TCP/IP. You must observe these restrictions if you wish to write CMX applications that are portable (with respect to station and computer interconnection). Note also that the transport systems for station interconnection and for computer interconnection using TCP/IP do not provide certain system and user options.

Restrictions associated with station interconnection

The ICMX(L) interface has the following functional restrictions when using CCP-STA V2.1 or later. These restrictions should also be observed by CMX applications via computer interconnection, if the goal is to write CMX applications that are portable (with respect to station and computer interconnection). These

restrictions stem from the fact that the actual transport system is either entirely absent (primary) or is located in the partner system.

t_conrq()

Each CMX application can maintain only one connection via station interconnection at a time, regardless of how it is organized into processes. However, a process may attach itself to several applications simultaneously, as is the case with computer interconnection. Reaction in case of faulty operation: T_DISIN with reason T_RLCONGEST (448).

CCP-STA1 does not allow for transfer of expedited data. If expedited data is selected in a t_conrq() call, this is automatically countermanded in t_concf().

No user data may be passed in the transparent transmission mode of CCP-STA1.

t_conrs()

No user data may be passed with the connection response (t_conrs()).

Reaction in case of faulty operation:

Beginning with CCP-STA1 V2.5: Error termination with error value T_WPARAMETER.

t_datarq()

When CCP-STA1/SDLC is chosen, the total length of a message (TSDU) is limited to one data unit (TIDU).

With CCP-STA1/MSV1 the total length of a message is limited to 3064 or 4088 bytes, depending on whether the data is to be transmitted with or without byte splitting. See the CCP-STA1 manual [4] for details.

The maximum length of a data unit (TIDU) can be queried dynamically using t_info() and depends on the version of CCP-STA1/MSV1:

CCP-STA1/MSV1 V2.5 and higher

3064/4088 bytes (see above) for CC with 1 Mb memory

With the (less frequently used) active MSV1 protocol variant (see CCP-STA1 manual [4]), t_info() always returns the maximum value (4088 bytes) for CCs with 1Mb memory even if connection properties permit only a data length of 3064 bytes.

With a TIDU size of 440 bytes at most 10 TIDUs may be chained using T_MORE to form a message. If TIDU and maximum message length are the same (3064 bytes/4088 bytes), use of T_MORE is forbidden. An application program should not make any specific assumptions as to the TIDU size, so that it will be able to handle both cases.

Reaction in case of faulty operation:

If T_MORE used illegally: T_DISIN with reason T_RLINPAR (470)

Otherwise: T_DISIN with reason T_RLINTIDU (465)

When linkage into the TRANSDATA network is involved, following connection setup, in which preferably the local CMX application should be the calling partner, the calling partner should also take the initiative in data exchange. Otherwise, heavy load in the attached processor may result in failure in the connection setup.

t_datain(), t_datastop()

Flow control exercised on the receiving side is purely local. It does not affect the partner's sending behavior. For this reason, data received on a connection and not fetched within a certain time period (about 1 minute) is discarded in order to permit other transfers on the line.

t_disrq(), t_disin()

During disconnection no user data may be passed.

Reaction in case of faulty operation:

Beginning with CCP-STA1 V2.5: Error termination with error value T_WPARAMETER.

Name space

From the point of view of the network, the possible names for CMX applications accessible via CCP-STA1 are limited by their generation as stations in the attached processor.

Restrictions associated with computer interconnection via RFC1006 over TCP/IP

The following functional restrictions apply when using computer interconnection via RFC1006 over TCP/IP:

t_attach()

With computer interconnection via RFC1006 over TCP/IP, two processes of an application cannot successfully attach themselves to CMX at the same time using T_PASSIVE.

If a second process of an application tries to attach itself to CMX using T_PASSIVE, t_attach() returns the value T_NOTFIRST.

With computer interconnection via RFC1006 over TCP/IP, this return value means that no connection requests will be delivered to this process.

If, when an application attempts to attach itself, another application with the same name already exists that attached itself without T_PASSIVE, the value T_OK is returned if the new application succeeds in attaching itself. In this case the value T_NOTFIRST is not returned, as is the case with the other interconnections.

t_conrq(), t_conrs(), t_concf(), t_conin()

The user option "Monitoring of inactive time" is not supported for connections via RFC1006 over TCP/IP. For this reason T_NO must be specified for the inactive time for the connection (t_timeout parameter in the t_conrq() or t_conrs() call).

The selection of expedited data is automatically countermanded internally so that T_NO is always returned for t_concf() and t_conin().

t_disrq(), t_detach(), t_disin()

When computer interconnection is implemented using RFC1006 over TCP/IP, an explicit or implicit disconnect does not overtake data sent previously. The disconnect is not delivered to a process until it has received all outstanding data, even that for which the process has blocked the indication with t_datastop().

t_datastop

If a process has blocked incoming data indication it cannot receive the events T_DISIN or T_DATAGO for the connection.

t_xdatgo(), t_xdatin(), t_xdatrq(), t_xdatstop()

(System option: exchange of expedited data) When two applications are connected via TCP/IP with RFC1006, no expedited data can be sent. If the CMX calls t_xdatgo(), t_xdatin(), t_xdatrq() and t_xdatstop() are used in the applications, CMX rejects them with (T_ERROR and) T_WSEQUENCE.

Restrictions associated with computer interconnection via CCP-WANx and CCP-ETHN

t_conin()

If the connection request (T_CONIN) is not answered immediately with t_conrs() or t_disrq(), the called TS application and calling TS application will receive T_DISIN with the disconnect reason T_UNKNOWN after a predefined period (which is CCP-dependent).

In the case of WAN-NEA or ETHN, if expedited data is sent three times and is not collected by the partner, the connection is cleared down.

9.4 System options and message length

It is important to note when creating TS applications that the system options to "exchange user data when setting up and closing down a connection" and to "exchange expedited data" are not supported by every

transport system (CCP profile). Moreover, in transport systems that do support these system options, the permitted length of the user data or the expedited data unit is not always identical.

In the case of station interconnections, the length of a message (TSDU) and the number of normal data units of a message (TIDUs) are also restricted.

Information on which CCP profiles support these system options and details on the supported length for user data, expedited data and messages may be obtained from the Release Notice.

9.4.1 Programming notes

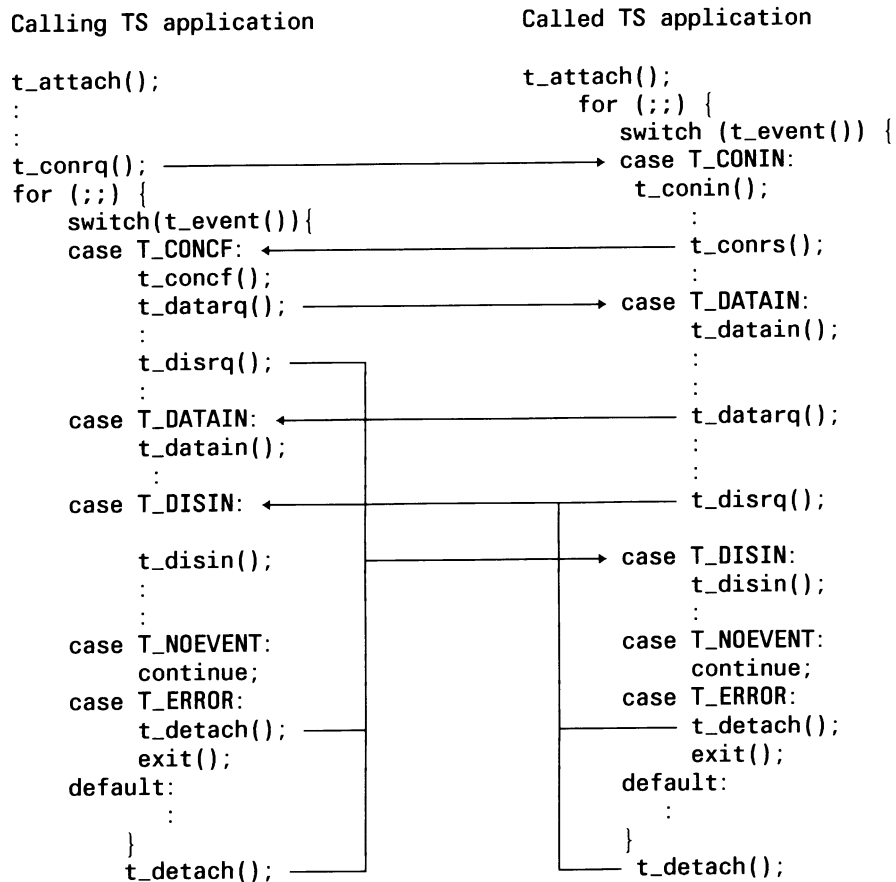
The primary purpose of ICMX(L) is to make TS applications independent of the transport systems used. This allows TS applications to execute in a variety of network environments. ICMX(L) supports this independence for TS applications that adhere to the following rules:

1. The application should make no explicit assumptions regarding the length of a TIDU or regarding the way TIDUs are packed for communication.
2. The limits defined in <cmx.h> for the options must never be exceeded. It is to be noted that some transport systems do not provide certain options.
3. The TS application should handle addressing exclusively with the aid of the TNSX; it should not construct any physical transport addresses in the programs.
4. CMX functions should not be called in signal handling routines; signal handling is not suitable for performing asynchronous processing outside the current context.
5. Function prototyping is supported by CMX. Thus, the program is notified whether or not the transferred parameter structure is correct during compilation.

ICMX is designed such that program runs can be event-driven. It is specially designed to allow an event loop to be programmed where special consideration is given to the individual events.

An event-driven design for two TS applications is shown in the following program example.

Example



9.5 Conventions

When using ICMX(L) the following conventions must be observed:

1. All identifiers starting with "_" (underscore) are reserved for the system software.
2. All identifiers starting with "t_" or "ts" or "Ts" are reserved for CMX.
3. All preprocessor definitions starting with "T_" or "TS_" are reserved for CMX.
4. At the request of the user, signals (usually SIGIO and/or SIGTERM) are sent (by CMX components in the kernel or socket components) and intercepted in the CMX library. User-defined signal routines should therefore be programmed with caution.

9.6 ICMX(L) - function calls

The following pages describe the CMX calls in detail. *Italic type in running text represents ordinary, replaceable formal parameters or the names of functions and files. Names in uppercase letters (e.g. T_MSGSIZE) represent constants that have been defined in a header file (with #define).*

The following conventions are used in the parameter descriptions:

->

Indicates a parameter in which CMX expects a value provided by the caller.

<-

Indicates a parameter in which CMX returns a value after the call.

<>

Indicates a parameter in which the caller must provide a value, which is then modified by CMX. Modification generally only takes place if processing was successful. If it was unsuccessful the value remains unchanged.

Of course, if a parameter involves a pointer, this marking does not refer to the pointer itself (which is always provided by the caller), but instead to the contents of the field to which the pointer points.

In all cases, for values to be returned by CMX appropriate storage space must be provided by the caller and a pointer must be passed to CMX.

9.6.1 t_attach - Attach a process to CMX (attach process)

t_attach() attaches the current process to CMX. The parameters passed in the t_attach() call specify:

- the TS application for which the process is being attached,
- the types of connection setup (passive, active) that are possible for the process in this TS application,
- the number of connections the process may have simultaneously in this TS application.

The TS application for which the process is being attached has a GLOBAL NAME and one or more T-selectors that are unique in the local system. The T-selectors combine to form the LOCAL NAME. The LOCAL NAME must be passed to CMX as a parameter of the t_attach() call. With the help of the call t_getloc() and the GLOBAL NAME of the TS application the LOCAL NAME can be queried from the TNSX and placed in a data area. A pointer to this data area is then passed in the t_attach() call.

Using repeated t_attach() calls, the current process may attach itself to CMX for several different TS applications.

Likewise, several different processes may attach themselves to CMX for the same TS application, i.e. using the same LOCAL NAME. The first process to attach itself for a TS application generates the TS application. CMX accepts connection requests for a TS application from the network as soon as a process of the TS application has attached itself to CMX for the acceptance of connection indications, i.e. when T_PASSIVE is specified in t_apmode.

If more than one process has attached itself for a TS application with T_PASSIVE, CMX initially delivers all connection indications for the TS application to the process that first attached itself for the TS application with T_PASSIVE. Only when the maximum number of connections that this process may have for the TS application is attained are arriving connection indications delivered to one of the other processes. The order in which this is done is not defined.

Notes

At the first t_attach(), several file descriptors are assigned in the running process. In the case of a computer interconnection (except with TCP/IP), one file descriptor remains assigned for the life of the process; if with TCP/IP, several file descriptors. For all interconnections, the file descriptors are released at the last t_detach() of the process.

T_OK with several T-selectors means that the attachment was successful for at least one T-selector.

```
#include <cmx.h>
```

```
int t_attach(name, opt)
struct t_myname *name;
t_opta *opt;
```

-> name

For name, specify a pointer to a structure t_myname with the LOCAL NAME of the TS application. The LOCAL NAME is returned by the TNSX as a property of the GLOBAL NAME of the TS application.

<> opt

For the parameter opt, specify the value NULL or a pointer to a union with user options. If opt = NULL is specified, CMX uses the given default values. The following structures are defined in <cmx.h>:

```

    struct t_opta1 {
->    int t_optnr;    /* Option no. */
->    int t_apmode;   /* Process mode */
->    int t_conlim;   /* Number of connections */
    }

    struct t_opta2 {
->    int t_optnr;    /* Option no. */
->    int t_apmode;   /* Process mode */
->    int t_conlim;   /* Number of connections */
->    int t_uattid;   /* User attachment reference */
<-    int t_attid;    /* CMX attachment reference */
<-    int t_ccbits;   /* Bit list of CCs affected */
<-    int t_sptypes;  /* Address formats affected */
    }

    struct t_opta5 {
->    int t_optnr;    /* Option no. */
->    int t_apmode;   /* Process mode */
->    int t_conlim;   /* Number of connections */
->    int t_uattid;   /* User attachment reference */
<-    int t_attid;    /* CMX attachment reference */
<-    int t_ccbits;   /* Bit list of CCs affected */
<-    int t_sptypes;  /* Address formats affected */
<-    int t_evref;    /* Reference point */
    }

    struct t_opta6 {
->    int t_optnr;    /* Option no. */
->    int t_apmode;   /* Process mode */
->    int t_conlim;   /* Number of connections */
->    int t_uattid;   /* User attachment reference */
<-    int t_attid;    /* CMX attachment reference */
<-    int t_cclist;   /* Address of the CC list */
<-    int t_sptypes;  /* Address formats affected */
    }

```

t_optnr

Option number. Specify:

T_OPTA1 in t_opta1

T_OPTA2 in t_opta2

T_OPTA5 in t_opta5

T_OPTA6 in t_opta6

t_apmode

t_apmode specifies the types of connection setup possible for the process in this TS application.

Permissible values are:

T_ACTIVE

The process is to actively set up connections.

T_PASSIVE

The process is to wait passively for requests to set up connections.

t_conlim

For t_conlim, specify the maximum number of simultaneous connections that the process may maintain in this TS application.

If t_conlim = T_NOLIMIT is specified, the process may maintain the installation-specific maximum number of simultaneous connections.

Otherwise, the following values may be specified:

for computer interconnection: `t_conlim > 0`
 for station interconnection: `t_conlim = 1`.

Default value if NULL specified: 1

`t_uattid`

In the field `t_uattid` you can pass CMX any user reference desired for this application. This user reference will be subsequently returned by CMX as an option in `t_event`, i.e. when the current process queries CMX regarding the arrival of an event.
 This user reference enables a process that controls multiple TS applications to more easily associate an arriving event with the appropriate attachment.
 Default value if NULL specified: 0

`t_attid`

This field serves trace and diagnostic purposes. It is used exclusively for logging.
 In the `t_attid` field CMX returns the CMX-internal reference to the attachment.

`t_ccbits`

This field serves trace and diagnostic purposes. It is used exclusively for logging.
 In `t_ccbits` CMX returns a bit-encoded list of the CCs for which the attachment was successful.
 The meaning of the list may be obtained from `<cmx.h>`.

`t_sptypes`

This field serves trace and diagnostic purposes. It is used exclusively for logging.
 In `t_sptypes` CMX returns a bit-encoded list of the address formats for which this attachment was successful.
 The meaning of the list may be obtained from `<cmx.h>`.

`t_evref`

Event reference point. This is permitted only for compatibility with CMX (BS2000) and is not supported in CMX (SINIX).

`t_cclist`

This pointer to a CC list is used for tracing and diagnostic purposes. It is used exclusively for logging.
 In `t_cclist`, CMX provides a pointer to a CC list of the address formats for which this attachment was successful.
 The values are explained in `<cmx.h>`.

Return values

`T_OK`

The call was successful. The process was the first to attach itself with this name.

`T_NOTFIRST`

The call was successful. However, the process was not the first to attach itself for this TS application.

`T_NOTPRIV`

The call was not successful. The non-privileged process attempted to connect to a privileged socket.
 This return value occurs only for attachments for TCP/IP.

`T_ERROR`

Error. Error code can be queried using `t_error()`.

Errors

If an error occurs the following error values are possible. They can be queried by calling `t_error()`.

For error type `T_CMXTYPE` and error class `T_CMXCLASS`, the following may occur:

`T_ENOENT`

All allocated resources are already occupied.

`T_EFAULT`

At least one of the pointers `name` or `opt` (`!= NULL`) does not point to the process address space.

`T_WPARAMETER`

The LOCAL name passed with `name` or one of the options specified in `opt` has an invalid format or contains illegal values.

`T_WAPPLICATION`

The process is already attached for the TS application specified in `name`, or the LOCAL NAME that was specified in `name` is already being used by an XTI process.

`T_WAPP_LIMIT`

The process has already attached itself for all applications available to it, or the maximum number of TS applications has been reached.

`T_WPROC_LIMIT`

The maximum number of processes that CMX can use has been reached.

`T_NOCCP`

No suitable CCP is available (at present) for the LOCAL NAME specified in `name`.

`T_WLIBVERSION`

The version of the CMX library linked into the process is incompatible with the operating system version.

See also

`t_detach()`, `t_event()`, `t_error()`, `t_getloc()`

Restrictions



The tcp port 102 is reserved for applications implementing the RFC1006. It is used by RFC1006 applications when waiting for incoming connection requests.

On UNIX systems this port is a privileged port. This port may only be used by applications which have access rights as superuser.

This implies for CMX applications which are working with RFC1006 over TCP/IP and waiting for incoming connections requests (parameter `t_apmode==T_PASSIVE` when calling `t_attach()`) that they also must run with access rights as superuser.



The tcp port 102 is used in CMX applications over RFC1006. A port number, which is used to receive tcp events, may be attached to only one application at a time. This implies for CMX application when using RFC1006 that the CMX function `t_attach()` with parameter `t_apmode==T_PASSIVE` may be called only once. If more than one CMX connection shall be used, the parameter `t_conlim` must be set to the appropriate value.

9.6.2 t_callback - Register a callback routine

Using `t_callback`, an application can attach its own function for event handling in ICMX(L), which are then called when `t_event` is executed. This function is referred to as a callback function, because it is called by an ICMX(L) function (`t_event()`) that belongs to the application.

The callback routine enables the application to use `t_event()` to simultaneously wait not only for TS events, but also for other events that are important for the application (e.g. terminal I/O). A prerequisite here is that the event sources can be described by means of file descriptors so that the callback routine can use the system functions `select()` or `poll()` (see UNIX Programmer's Reference Guide) for checking events.

The callback routine is used as follows: when routine is called, the function `t_event()` passes bit lists containing file descriptors, which are used internally by ICMX(L) and for which events are outstanding. The transfer is the same as for the `select()` call. The function routine can now add application-specific file descriptors and can call the function `select()` with the completed bit lists. This makes it possible to simultaneously wait for TS events and other application-specific events. If an event occurs, routine must check whether or not the event was application-specific. If so, routine ends with `T_USEREVENT`; otherwise, it ends with `T_TSEVENT`. Before routine terminates, the routine can continue processing an application-specific event. However, routine may never enter a wait state that cannot be interrupted by signals.

The behavior of `t_event` depends on the return value of routine. If a TS event has not occurred, `t_event()` returns to the application with `T_NOEVENT`. Otherwise, the TS event is displayed.

A callback routine enables the application to optimize event handling according to your own criteria. If it is not implemented correctly, the handling of TS events by `t_event()` may not be reliable. You should therefore follow the implementation guidelines listed below.

The functional sample of routine is similar to the function `select()` and looks like this:

```
#include <sys/select.h>
#include <cmx.h>

int routine (fdsetsize, rfds, wfds, xfds, time, usr)
int fdsetsize;
fd_set *rfds, *wfds, *xfds;
struct timeval *time;
void *usr;
```

-> `fdsetsize`

Number of file descriptors that are used by ICMX(L) internally.

<> `rfds`

Pointer to a bit list of file descriptors for which ICMX(L) expects read events internally. The return list must contain at least the file descriptors of ICMX(L) for which a read event has occurred. File descriptors that are not affected are ignored in the list.

<> `wfds`

Pointer to a bit list of file descriptors for which ICMX(L) expects write permission internally. The return list must contain at least the file descriptors of ICMX(L) for which this type of event has occurred. File descriptors that are not affected are ignored in the list.

<> `xfds`

Pointer to a bit list of file descriptors for which ICMX(L) expects exceptional conditions to occur. The return list must contain at least the file descriptors of ICMX(L) for which this type of event has occurred. File descriptors that are not affected are ignored in the list.

-> time

specifies the maximum time spent waiting for an event to occur. The value 0 indicates that routine can check whether an event exists, but cannot enter a wait state. The value -1 means that routine should wait for the event for an unlimited period of time. The time value is derived from the `t_timeout` value when `t_event()` is called.

Please note that routine must support the value 0 at all times, even if the application never calls `t_event()` in `T_CHECK` mode.

-> usr

Pointer that was passed to `t_callback` by the application when routine was attached (see below). The contents are not checked by `ICMX(L)`. The application can pass application-specific information to routine via `usr`.

Return value

T_NOEVENT

Neither a user event nor a TS event has occurred within time. The function was interrupted by a signal or an internal error occurred.

In this case, `t_event()` also terminates with `T_NOEVENT`.

T_TSEVENT

A TS event has occurred. `t_event()` then checks which TS event occurred and terminates with this event.

If `t_event()` cannot find a TS event, it terminates with `T_NOEVENT`.

T_USEREVENT

An application-specific event has occurred. `t_event()` terminates either with `T_NOEVENT` if `t_event()` does not detect a TS event, or it reports the TS event that `t_event()` detected outside routine.

Errors

If routine has to terminate prematurely because of an internal error, it must terminate with `T_NOEVENT`. You should take note of the error status internally, so that the application can take appropriate action when `t_event()` has terminated.

Implementation guidelines

- The `ICMX(L)`-specific file descriptors may not be distorted by routine. TS events can only be identified if these file descriptors are passed to `select()` or `poll()`. If the file descriptors are distorted while routine is running, `t_event()` is no longer reliable.
- The callback routine may only enter wait states that can be interrupted by all signals.
- The callback routine may not call `t_event()`. `t_event()` rejects the recursive call with `T_CBRECURSIVE`.
- The callback routine is UNIX-specific, as it explicitly uses file descriptors. The concept is therefore not provided in `ICMX(L)` implementations in BS2000 and MS-DOS.

Information on the callback routine is reported to `ICMX(L)` with `t_callback`:

```
#include <cmx.h>

int (*t_callback(routine, usr, opt))()
int (*routine)();
void *usr;
void *opt;
```

-> routine

Pointer to the callback routine that is to be called by `t_event()`. A callback routine that was attached to ICMX(L) is detached again with NULL.

-> usr

Pointer to an application-specific data area that is not checked by ICMX (L). The pointer is passed to the callback routine by `t_event()` when this is called.

-> opt

Reserved for future extensions. The value must be NULL.

Return value

T_OK

The pointer to the old callback routine is returned. The return value "NULL" means that a callback routine was not included.

T_ERROR

Error. Error code can be queried with `t_error()`.

Errors

In the event of an error the following error values are possible. They can be queried by calling `t_error()`.

The following can occur for error type T_CMXTYPE and error class T_CMXCLASS:

T_WPARAMETER

opt is not NULL.

T_WSEQUENCE

The process has not been attached in any TS application.

See also

`t_event()`

9.6.3 t_concf - Establish connection (connect confirmation)

t_concf() accepts a T_CONCF event from CMX previously reported with t_event(). T_CONCF indicates that the called TS application has positively answered a connection request (t_conrq() call) of the current process.

t_concf() returns:

- The user data that the called TS application included, if the transport system used provides this option.
- The answer of the called TS application if the current process proposed the exchange of expedited data when issuing the connection request t_conrq().

If the t_concf() call is successful the connection is established for the current process. As soon as a connection is established, the TS application (not CMX) has the initiative. It may:

- send normal data and (if agreed) expedited data, or
- indicate, through t_event(), that it is ready to receive normal data or (if agreed) expedited data or close down the connection.

```
#include <cmx.h>
```

```
int t_concf(tref,opt)
```

```
int *tref;
```

```
t_opt1 *opt;
```

-> tref

Pointer to a field with the transport reference of the connection, passed to the current process via t_event().

<> opt

For opt, specify the value NULL or a pointer to a union containing a structure with system options. This union is used to receive the user data that the called TS application included with its answer to the connection request.

If opt = NULL is specified, CMX discards the user data and options.

If the called TS application specified no user data and no options, CMX uses the given default values.

The following structure is defined in <cmx.h>:

```
struct t_optc1 {
->  int  t_optnr;      /* Option no. */
<-  char *t_udatap;   /* Data buffer */
<>  int  t_udatal;    /* Length of the data buffer */
<-  int  t_xdata;     /* Choice for expedited data */
<-  int  t_timeout;   /* Inactive time */
};
```

t_optnr

Option number. Specify T_OPTC1.

t_udatap

Pointer to a data area in which CMX enters the user data received from the called TS application.

Default value if NULL specified: Undefined

t_udatal

Prior to the call 0 or the length of the data area t_udatap must appear here.
 The area must be large enough that the received data completely fits. The maximum length required depends on the transport system used. T_MSG_SIZE is the maximum size suitable for all transport systems. T_MSG_SIZE is defined in <cmx.h>.
 After the call, CMX returns in this field the number of bytes placed in t_udatap.
 Default value if NULL specified: 0

t_xdata

CMX returns here the answer of the called TS application if the exchange of expedited data was proposed at connection setup. The answer is binding. Possible answers:

T_YES

The called TS application accepts the proposal.

T_NO

The called TS application rejects the proposal.

Default value if NULL specified: T_NO

t_timeout

This field always contains T_NO.

Return values**T_OK**

The call was successful.

T_ERROR

Error. Error code can be queried using t_error().

Errors

If an error occurs the following error values are possible. They can be queried by calling t_error().

For error type T_CMXTYPE and error class T_CMXCLASS, the following may occur:

T_EFAULT

At least one of the pointers opt (!= NULL) or t_udatap (!= NULL and t_ndatal != 0) does not point to the process address space.

T_WSEQUENCE

The process is not attached for any TS application, or no T_CONCF was indicated on the connection specified by tref.

T_WPARAMETER

The options specified in opt have an invalid format or contain illegal values, or the buffer for the data to be received is too small.

T_CCP_END

The CCP is no longer operational.

In addition, the errors listed under ioctl(2) may occur.

See also

t_conrq(), t_error(), t_event()

9.6.4 t_conin - Receive connection request (connect indication)

t_conin() accepts a T_CONIN event previously reported with t_event(). T_CONIN indicates that a calling TS application wishes to set up a connection to the current process.

The call returns:

- the TRANSPORT ADDRESS of the calling TS application,
- the LOCAL NAME of the local TS application, and
- the user data that the calling TS application included.

Subsequently the connection request may be answered (confirmed) with t_conrs() or rejected with t_disrq().

```
#include <cmx.h>
```

```
int t_conin(tref, toaddr, fromaddr, opt)
int *tref;
union t_address *toaddr;
union t_address *fromaddr;
t_opt1 *opt;
```

-> tref

Pointer to a field with the transport reference of the connection, passed to the current process via t_event().

<- toaddr

Pointer to a union t_address in which CMX returns the LOCAL NAME of the local TS application that is to receive the connection.

If the current process is attached for multiple TS applications, with the aid of this information the connection request can be associated with the correct TS application.

<- fromaddr

Pointer to a union t_address in which CMX returns the TRANSPORT ADDRESS of the calling TS application. The TRANSPORT ADDRESS can be converted to the GLOBAL NAME of the calling TS application with the aid of the call t_getname().

Note

If an RFC1006 type TRANSPORT ADDRESS is received during communication via RFC1006 over TCP/IP, the address may not be passed in binary form to t_conrq() in a subsequent active connection setup, because the internal address components that are evaluated by t_conrq() are missing. The application must query the GLOBAL NAME using t_getname(), and then query the address again using t_getaddr().

<> opt

For opt, specify the value NULL or a pointer to a union containing a structure with system options.

This union is used to fetch the user data that the calling TS application specified at connection setup. If opt = NULL is specified, CMX discards the user data and options.

If the calling TS application specified no user data and no options in t_conrq(), CMX returns the specified default values.

The following structure is defined in <cmx.h>:

```

    struct t_optc1 {
->    int    t_optnr;      /* Option no. */
<-    char  *t_udatap;    /* Data buffer */
<>    int    t_udatal;     /* Length of the data buffer */
<-    int    t_xdata;     /* Choice for expedited data */
<-    int    t_timeout;   /* Inactive time */
    };

```

t_optnr

Option number. Specify T_OPTC1.

t_udatap

Pointer to a data area in which CMX enters the user data received from the calling TS application.

Default value if NULL specified: Undefined

t_udatal

Prior to the call 0 or the length of the data area t_udatap must appear here.

The area must be large enough that the received data completely fits. The maximum length required depends on the transport system used. T_MSG_SIZE is the maximum size suitable for all transport systems. T_MSG_SIZE is defined in <cmx.h>.

After the call, CMX returns in this field the number of bytes placed in t_udatap.

Default value if NULL specified: 0

t_xdata

In this field CMX returns the proposal of the calling TS application regarding expedited data.

Possible answers:

T_YES

The calling TS application proposes exchanging expedited data.

T_NO

The exchange of expedited data is ruled out by the calling TS application.

If the calling TS application proposes exchanging expedited data (T_YES), the answer of the current process in the subsequent t_conrs() is final.

If the calling TS application desires no expedited data (T_NO), none can be requested by the current process in the subsequent t_conrs(). It may then be necessary for the current process to reject the connection request with t_disrq().

Default value if NULL specified: T_NO

t_timeout

This field always contains T_NO.

Return values

T_OK

The call was successful.

T_ERROR

Error. Error code can be queried using t_error().

Errors

If an error occurs the following error values are possible. They can be queried by calling t_error().

For error type T_CMXTYPE and error class T_CMXCLASS, the following may occur:

T_EFAULT

At least one of the pointers toaddr, fromaddr, opt (!= NULL) or t_udatap (!= NULL and t_ndatal != 0) does not point to the process address space.

T_WSEQUENCE

The process is not attached for any TS application, or no T_CONIN was indicated on the connection specified by tref.

T_WPARAMETER

The options specified in opt have an invalid format or contain illegal values, or the buffer for the data to be received is too small.

T_CCP_END

The CCP is no longer operational.

In addition, the errors listed under ioctl(2) may occur.

See also

t_attach(), t_conrs(), t_conrq(), t_disrq(), t_error(), t_event(), t_getname()

9.6.5 t_conrq - Request connection (connection request)

t_conrq() requests the establishment of a transport connection from the local TS application to a called TS application (active connection setup).

More specifically, the effects of t_conrq() are:

- The called TS application receives the event T_CONIN as a connection indication, to which it must respond.
The answer of the called TS application is later indicated to the current process by CMX in a t_event() call as event T_CONCF or T_DISIN.
- The called TS application may be sent user data along with the connection request, if the transport system used provides this option.

```
#include <cmx.h>
```

```
int t_conrq(tref, toaddr, fromaddr, opt)
int *tref;
union t_address *toaddr;
union t_address *fromaddr;
t_opt13 *opt;
```

<- tref

Pointer to a field in which CMX returns the connection-specific transport reference. This uniquely identifies the connection in the subsequent communication phases. It must therefore be specified with all calls that involve this connection.

-> toaddr

Pointer to a union t_address with the TRANSPORT ADDRESS of the called TS application. The TRANSPORT ADDRESS is returned by the TNSX as a property of the GLOBAL NAME of the called TS application. It can be ascertained from the TNSX using a t_getaddr() call.

Note

In the case of connection setup via STA1, WAN or LAN interconnection CMX internally examines the ROUTING INFORMATION property of the called TS application. If this property is not present or if there is more than one TS application with the called TRANSPORT ADDRESS toaddr, a default value is used. The effect of this is that the connection is set up via the first suitable transport system that the called TRANSPORT ADDRESS fits.

-> fromaddr

For fromaddr, specify a pointer to a union t_address with the LOCAL NAME of the calling TS application. The same LOCAL NAME must be specified here as was specified in t_attach() for this TS application.

-> opt

For opt, specify the value NULL or a pointer to a union with system options.
This is used to specify the user data and options that the called TS application is to receive with the connection indication.
If opt = NULL is specified, CMX uses the given default values.

The following structures are defined in <cmx.h>:

```

    struct t_optc1 {
->    int    t_optnr;      /* Option no. */
->    char  *t_udatap;    /* Data buffer */
->    int    t_udatal;    /* Length of the data buffer */
->    int    t_xdata;     /* Choice for expedited data */
->    int    t_timeout;   /* Inactive time */
    };

    struct t_optc3 {
->    int    t_optnr;      /* Option no. */
->    char  *t_udatap;    /* Data buffer */
->    int    t_udatal;    /* Length of the data buffer */
->    int    t_xdata;     /* Choice for expedited data */
->    int    t_timeout;   /* Inactive time */
->    int    t_ucepid;    /* User connection reference */
    };

```

t_optnr

Option number. Specify:

T_OPTC1 in t_optc1

T_OPTC3 in t_optc3

t_udatap

Pointer to a storage area containing user data that the called TS application is to receive with the connection indication.

Default value if NULL specified: Undefined

t_udatal

Length of the user data, in bytes, to be transferred from the area t_udatap.

If 0 is specified for t_udatal, t_udatap is ignored.

The maximum value for t_udatal depends on the transport system (see the Release Notice).

Default value if NULL specified: 0

t_xdata

In the t_xdata parameter the current process informs the called TS application as to whether it is ready to exchange expedited data.

Permissible values are:

T_YES

Exchange of expedited data proposed.

T_NO

Exchange of expedited data ruled out.

Default value if NULL specified: T_NO

t_timeout

For t_timeout, specify the inactive time for the connection. The inactive time specifies how long the connection may be inactive before it will be closed down by CMX. It begins only when all data has been retrieved.

Possible specifications:

T_NO

The inactive time of the connection will not be monitored.

n > 0

The connection may be inactive for n seconds. Thereafter CMX will close it down.

Default value if NULL specified: T_NO.

t_ucepid

This field can be used to pass a freely-selectable user reference for this connection to CMX. This user reference can be returned to the current process by CMX as an option in a `t_event()` call.

If the current process is maintaining multiple connections this mechanism enables it to associate a TS event with the appropriate connection via a user-defined attribute. The user reference constitutes an alternative to the transport reference `tref`, defined by CMX.

Default value if NULL specified: 0

Return values**T_OK**

The call was successful.

T_ERROR

Error. Query error code using `t_error()`.

Errors

If an error occurs the following error values are possible. They can be queried by calling `t_error()`.

For error type `T_CMXTYPE` and error class `T_CMXCLASS`, the following may occur:

T_EFAULT

At least one of the pointers `toaddr`, `fromaddr`, `opt` (`!= NULL`) or `t_udatap` (`!= NULL` and `t_ndatal` `!= 0`) does not point to the process address space.

T_WSEQUENCE

The process is not attached for any TS application, or the process has not set `T_ACTIVE` in `t_apmode` for the local TS application, specified in `fromaddr`.

T_WPARAMETER

The TRANSPORT ADDRESS passed with `toaddr` or the LOCAL name passed with `fromaddr` or one of the options specified in `opt` has an invalid format or contains illegal values.

T_WAPPLICATION

The process is not attached for the TS application that has the LOCAL NAME passed with `fromaddr`.

T_WCONN_LIMIT

The process has already used the number of connections specified for this TS application in `t_attach()` (`t_conlim` parameter) or the system limit for connections has been exceeded.

T_NOCCP

The TRANSPORT ADDRESS specified in `toaddr` is not supported by any (currently) operational CCP or the LOCAL NAME specified in `fromaddr` contains no information for this CCP.

T_ETIMEOUT

The CCP does not respond in the time limit.

T_CCP_END

The CCP is no longer operational.

In addition, the errors listed under `ioctl(2)` may occur.

See also

`t_attach()`, `t_error()`, `t_event()`, `t_getaddr()`

9.6.6 t_conrs - Respond to connection request (connection response)

t_conrs() is used by the called TS application to accept (confirm) the connection request of a calling TS application, the connection request having been previously indicated to the current process in t_event(), with the event T_CONIN. The current process must accept the T_CONIN event with t_conin() (passive connection setup) before calling t_conrs(). The calling TS application receives this response as connection confirmation with the event T_CONCF.

With t_conrs()

- information can be sent to the calling TS application, if the transport system used provides this option;
- the connection is completely set up for the current process.

As soon as a connection has been established, the TS application (not CMX) has the initiative. It may:

- send both normal data and (if agreed) expedited data, or
- indicate, via t_event(), that it is prepared to receive normal data or (if agreed) expedited data, or
- close down or redirect the connection.

```
#include <cmx.h>
```

```
int t_conrs(tref,opt)
int *tref;
t_opt13 *opt;
```

-> tref

Pointer to a field with the transport reference for the connection used in the corresponding t_conin().

-> opt

For opt, specify the value NULL or a pointer to a union with system options.

This is used by the current process to pass the user data that the calling TS application is to receive with the response to the connection request. If opt = NULL is specified, CMX uses the given default values.

The following structures are defined in <cmx.h>:

```
struct t_optc1 {
->  int t_optnr;      /* Option no. */
->  char *t_udatap;   /* Data buffer */
->  int t_udatal;     /* Length of the data buffer */
->  int t_xdata;      /* Choice for expedited data */
->  int t_timeout;    /* Inactive time */
};

struct t_optc3 {
->  int t_optnr;      /* Option no. */
->  char *t_udatap;   /* Data buffer */
->  int t_udatal;     /* Length of the data buffer */
->  int t_xdata;      /* Choice for expedited data */
->  int t_timeout;    /* Inactive time */
->  int t_ucepid;     /* User connection reference */
};
```

t_optnr

Option number. Specify:

T_OPTC1 in t_optc1

T_OPTC3 in t_optc3

t_udatap

Pointer to a storage area containing user data that the calling TS application is to receive.

Default value if NULL specified: Undefined

t_udatal

Length of the user data, in bytes, to be transferred from the area t_udatap.

If 0 is specified for t_udatal, t_udatap is ignored.

The maximum value for t_udatal depends on the transport system (see the Release Notice).

Default value if NULL specified: 0

t_xdata

In t_xdata the current process responds to the proposal of the calling TS application regarding the exchange of expedited data. The proposal was passed to the process via the t_conin() call.

Permissible values are:

T_YES

The proposal of the calling TS application regarding expedited data is accepted.

T_NO

Expedited data is refused.

The response is binding.

If the calling TS application had ruled out the use of expedited data, the response here must be T_NO.

Default value if NULL specified: T_NO

t_timeout

For t_timeout, specify the inactive time for the connection. The inactive time specifies how long the connection may be inactive before it will be closed down by CMX.

T_NO

Inactive time will not be monitored.

n > 0

The connection may be inactive for n seconds. Thereafter CMX will close it down.

Default value if NULL specified: T_NO.

t_ucepid

This field can be used to pass a freely-selectable user reference for this connection to CMX. This user reference can be returned to the current process by CMX as an option in a t_event() call.

If the current process is maintaining multiple connections this mechanism enables it to associate a TS event with the appropriate connection via a user-defined attribute. The user reference constitutes an alternative to the transport reference tref, defined by CMX.

Default value if NULL specified: 0

Return values

T_OK

The call was successful.

T_ERROR

Error. Query error code using t_error().

Errors

If an error occurs the following error values are possible. They can be queried by calling `t_error()`.

For error type `T_CMXTYPE` and error class `T_CMXCLASS`, the following may occur:

`T_EFAULT`

At least one of the pointers `opt` (`!= NULL`) or `t_udatap` (`!= NULL` and `t_ndatal != 0`) does not point to the process address space.

`T_WSEQUENCE`

The process is not attached for any TS application, or the call was not preceded by a successful `t_conin()` call.

`T_WPARAMETER`

The options specified in `opt` have an invalid format or contain illegal values.

`T_COLLISION`

The event `T_DISIN` (disconnect indication) has arrived for the connection, but has not yet been fetched with `t_event()`.

Response: Call `t_event()`.

`T_CCP_END`

The CCP is no longer operational.

In addition, the errors listed under `ioctl(2)` may occur.

See also

`t_conin()`, `t_error()`, `t_event()`

9.6.7 t_datago - Release the flow of data (data go)

t_datago() releases the blocked flow of data on the specified connection. By means of this call the current process informs CMX that it is again ready to receive data. This call also releases the flow of expedited data (if it is being used) if it (also) had been blocked. In particular, the effects of the call are:

- The current process can again receive the events T_DATAIN and T_XDATIN for the specified connection, if they are waiting.
- The sending TS application receives the event T_DATAGO. It may again send data.

Once a connection has been set up, the initiative rests with the TS application (not with CMX). It may send normal data and (if agreed) expedited data, or indicate, with t_event(), that it is ready to receive normal data or (if agreed) expedited data.

```
#include <cmx.h>
```

```
int t_datago(tref)
int *tref;
```

-> tref

Pointer to a field with the transport reference of the connection on which the flow of data is to be released.

Return values

T_OK

The call was successful.

T_ERROR

Error. Query error code using t_error().

Errors

If an error occurs the following error values are possible. They can be queried by calling t_error().

For error type T_CMXTYPE and error class T_CMXCLASS, the following may occur:

T_WSEQUENCE

The process is not attached for any TS application, or the process is not in the data phase for the connection specified in tref, or the flow of data has not been blocked.

T_CCP_END

The CCP is no longer operational.

In addition, the errors listed under ioctl(2) may occur.

See also

t_datastop(), t_xdatstop(), t_error(), t_event(), t_redin()

9.6.8 t_datain - Receive data (data indication)

t_datain() accepts a T_DATAIN event previously reported via t_event(). The t_datain() call must be made before the next t_event().

By means of this call the current process receives data of a Transport Interface Data Unit (TIDU) belonging to the current Transport Service Data Unit (TSDU) from the sending TS application on the specified connection.

The maximum length of a TIDU depends on the transport system used. It can be queried for a connection that has already been set up by means of t_info().

A TIDU need not be completely full. The breakdown of a TSDU into TIDUs is purely local and does not indicate anything regarding the breakdown of the TSDU into TIDUs at the sending TS application. Between two TIDUs of a TSDU any other CMX events can occur for the same or a different connection.

When t_datain() is called a contiguous data area datap is provided in which CMX enters the data of the TIDU received.

t_datain() indicates:

- (in the chain parameter)

whether a further TIDU belonging to the current TSDU exists (chain=T_MORE) or does not exist (chain = T_END).

The individual TIDUs of a TSDU are each indicated via t_event() with the event T_DATAIN.

- (with the return value)

whether the current TIDU has been completely read or not.

If the value T_OK is returned, the TIDU has fit into the storage area provided. The current process has completely received the current TIDU.

If a value n > 0 is returned, only a part of the TIDU has been read. n is the number of bytes of the TIDU that have not yet been read (remaining length). In this case t_datain() or t_vdatain() must be called repeatedly until the entire TIDU has been read. Only then can other CMX calls be issued again, e.g. t_event().

```
#include <cmx.h>
```

```
int t_datain(tref, datap, datal, chain)
int *tref;
char *datap;
int *datal;
int *chain;
```

-> tref

Pointer to a field containing the transport reference of the connection, obtained via t_event().

<- datap

Pointer to a storage area in which CMX enters the data of the TIDU received.

<> datal

Prior to the call, for datal a pointer must be specified to a field in which the length of datap must be entered (at least 1). Following the call, CMX returns in this field the number of bytes entered in the storage area datap. This need not be the maximum length of the TIDU.

<- chain

chain is a pointer to a field in which CMX returns an indicator. This indicator shows whether or not an additional TIDU belonging to the TSDU exists.

Possible values:

T_MORE

Another TIDU belonging to the TSDU follows. It will be indicated with a separate T_DATAIN event.

T_END

The present TIDU is the last of the TSDU.

Return values

T_OK

The call was successful. The TIDU was completely read.

$n > 0$

n bytes remain from the TIDU.

T_ERROR

Error. Query error code using `t_error()`.

Errors

If an error occurs the following error values are possible. They can be queried by calling `t_error()`.

For error type T_CMXTYPE and error class T_CMXCLASS, the following may occur:

T_EFAULT

The pointer `datap` does not point to the process address space.

T_WSEQUENCE

The process is not attached for any TS application, or no T_DATAIN was indicated for the connection specified in `tref`.

T_WPARAMETER

The length specified in `data1` is invalid.

T_COLLISION

The event T_DISIN (disconnect indication) has arrived for the connection, but has not yet been fetched with `t_event()`.
Response: Call `t_event()`.

T_CCP_END

The CCP is no longer operational.

In addition, the errors listed under `ioctl(2)` may occur.

See also

`t_error()`, `t_event()`, `t_info()`

9.6.9 t_datarq - Send data (data request)

t_datarq() sends the next (or only) Transport Interface Data Unit (TIDU) of a Transport Service Data Unit (TSDU) to the receiving TS application on the specified connection.

The TIDU to be sent by t_datarq() must be provided by the current process in a contiguous data area.

If the TSDU is longer than one TIDU, it must be transferred using several t_datarq() calls in succession. Therefore in each t_datarq() call the sending process must specify in the chain parameter whether additional TIDUs belonging to the same TSDU follow.

The maximum length of a TIDU depends on the transport system used. It can be queried for an established connection by means of t_info().

If t_datarq() returns the value T_DATASTOP, the TIDU has been accepted by CMX but the flow of TIDUs on this connection has been blocked.

The flow of TIDUs can be blocked by:

- the receiving TS application,
which can block the flow of TIDUs by calling t_datastop() or t_xdatstop(), or
- CMX,
if the local buffer is full.

If the flow of TIDUs is blocked, before further TIDUs can be sent you must wait, by means of t_event(), for the event T_DATAGO for the connection.

Successful termination of t_datarq() (T_OK) does not mean that the receiving TS application has already accepted the data.

Unsuccessful termination of t_datarq() (T_ERROR) always means that an error has been detected locally.

```
#include <cmx.h>
```

```
int t_datarq(tref, datap, datal, chain)
int *tref;
char *datap;
int *datal;
int *chain;
```

-> tref

Pointer to a field with the transport reference of the connection.

-> datap

Pointer to a data area containing the TIDU to be sent.

-> datal

Pointer to a field containing the number of bytes to be sent from the storage area datap. At least 1 and at most the length of a TIDU must be specified.

-> chain

Pointer to an indicator used by the process to indicate whether there is an additional TIDU belonging to the TSDU.

Possible values:

T_MORE

Another TIDU belonging to the TSDU follows.

T_END

The present TIDU is the last of the TSDU.

Return values**T_OK**

The call was successful; further TIDUs may be sent immediately.

T_DATASTOP

The call was successful, but further TIDUs may not be sent until the event T_DATAGO has arrived for this connection.

T_ERROR

Error. Query error code using `t_error()`.

Errors

If an error occurs the following error values are possible. They can be queried by calling `t_error()`.

For error type `T_CMXTYPE` and error class `T_CMXCLASS`, the following may occur:

T_EFAULT

The pointer `datap` does not point to the process address space.

T_WSEQUENCE

The process is not attached for any TS application, or the process is not in the data phase for the connection specified in `tref`, or the flow of data is blocked.

T_WPARAMETER

The length specified in `datal` or the value specified in `chain` is invalid.

T_COLLISION

The event `T_DISIN` (disconnect indication) has arrived for the connection, but has not yet been fetched with `t_event()`.

Response: Call `t_event()`.

T_CCP_END

The CCP is no longer operational.

In addition, the errors listed under `ioctl(2)` may occur.

See also

`t_datastop()`, `t_error()`, `t_event()`, `t_info()`, `t_xdatstop()`

9.6.10 t_datastop - Stop the flow of data (data stop)

t_datastop() blocks the flow of data on the specified connection.

In particular, the effects of t_datastop() are:

- The current process tells CMX that, until further notice, it is not ready to receive data for this connection. However, a T_DATAIN event that has already been indicated must be responded to first.
- The current process no longer receives the event T_DATAIN for the specified connection. However, while the data flow is blocked it may call other CMX functions, e.g. to set up, close down or redirect an additional connection.
- The sending TS application receives the return value T_DATASTOP when it calls t_datarq(). It may not send any more data.

The flow of data is released with t_datago().

Expedited data is not affected by t_datastop().

```
#include <cmx.h>

int t_datastop(tref)
int *tref;
```

-> tref

Pointer to a field with the transport reference of the connection.

Return values

T_OK

The call was successful.

T_ERROR

Error. Query error code using t_error().

Errors

If an error occurs the following error values are possible. They can be queried by calling t_error().

For error type T_CMXTYPE and error class T_CMXCLASS, the following may occur:

T_WSEQUENCE

The process is not attached for any TS application, or the process is not in the data phase for the connection specified in tref, or a TIDU or an ETSDU has not yet been completely read.

T_CCP_END

The CCP is no longer operational.

In addition, the errors listed under ioctl(2) may occur.

See also

t_datarq(), t_datago(), t_event(), t_xdatstop()

9.6.11 t_detach - Detach a process from a TS application (detach process)

t_detach() detaches the current process for the TS application specified in the parameter name. If connections still exist for this process, they are implicitly closed down. Normally though, all connections for this process should be closed down with t_disrq() before calling t_detach().

When the last process of a TS application detaches itself, the TS application ceases to exist. Connection requests for that TS application will then no longer be accepted.

```
#include <cmx.h>

int t_detach(name)
struct t_myname *name;
```

-> name

Pointer to a structure t_myname with the LOCAL NAME of the TS application. The same LOCAL NAME is to be specified as was specified with t_attach().

Return values

T_OK

The call was successful.

T_ERROR

Error. Query error code using t_error().

Errors

If an error occurs the following error values are possible. They can be queried by calling t_error().

For error type T_CMXTYPE and error class T_CMXCLASS, the following may occur:

T_EFAULT

The pointer name does not point to the process address space.

T_WSEQUENCE

The process is not attached for any TS application.

T_WPARAMETER

The LOCAL name passed with name has an invalid format or contains illegal values.

T_WAPPLICATION

The process is not attached for the TS application that has the LOCAL NAME passed via name.

In addition, the errors listed under ioctl(2) may occur.

See also

t_attach(), t_error()

9.6.12 t_disin - Accept disconnection (disconnection indication)

t_disin() accepts a T_DISIN event previously reported with t_event(). T_DISIN indicates that the connection has been closed down.

t_disin() specifies whether the remote TS application or CMX initiated the T_DISIN event.

In addition, t_disin() returns:

- the user data sent by the remote TS application, if the T_DISIN event was initiated by the remote TS application and if the transport system used provides this option;
- the reason for closing the transport connection, if the T_DISIN event was initiated by CMX or by the transport system.

The reason for the disconnection is returned by t_disin() in hexadecimal form. The readable text form of the code can be obtained with the aid of t_preason() or t_strreason().

```
#include <cmx.h>
```

```
int t_disin(tref, reason, opt)
int *tref;
int *reason;
t_opt2 *opt;
```

-> tref

Pointer to a field containing the transport reference of the connection.

<- reason

Pointer to a field in which CMX enters the reason for the disconnection.

Possible values:

T_USER

The connection was closed down by the remote TS application.

other

The connection was closed down by CMX or the transport system. The possible values for this parameter and their meanings can be found in the appendix of this manual.

The code returned by CMX for the disconnection can be decoded with the aid of the cmxdec command.

<> opt

For opt, specify the value NULL or a pointer to a union containing a structure with system options. This union can be used to check the user data that the remote TS application specified when closing down the connection.

If opt = NULL is specified, CMX discards the user data and options.

If the remote TS application specified no user data and no options, CMX returns the default values specified.

The following structure is defined in <cmx.h>:

```
struct t_optc2 {
->  int  t_optnr;      /* Option no. */
<-  char *t_udatap;   /* Data buffer */
<>  int  t_udatal;    /* Length of the data buffer */
};
```

t_optnr

Option number. Specify T_OPTC2.

t_udatap

Pointer to a data area in which CMX enters the user data received from the remote TS application.

Default value if NULL specified: Undefined

t_udatal

Prior to the call 0 or the length of the data area t_udatap must appear here.

The area must be large enough that the received data completely fits. The maximum permissible length for the user data depends on the transport system used. T_MSG_SIZE is the maximum size suitable for all transport systems. After the call, CMX returns in this field the number of bytes placed in t_udatap.

Default value if NULL specified: 0

Return values**T_OK**

The call was successful.

T_ERROR

Error. Query error code using t_error().

Errors

If an error occurs the following error values are possible. They can be queried by calling t_error().

For error type T_CMXTYPE and error class T_CMXCLASS, the following may occur:

T_EFAULT

The pointer opt (!= NULL) does not point to the process address space.

T_WSEQUENCE

The process is not attached for any TS application, or no T_DISIN was indicated for the connection specified in tref.

T_WPARAMETER

The options specified in opt have an invalid format or contain illegal values, or the buffer for the data to be received is too small.

T_CCP_END

The CCP is no longer operational.

In addition, the errors listed under ioctl(2) may occur.

See also

t_detach(), t_disrq(), t_event(), t_preason(), t_streason()

9.6.13 t_disrq - Close down connection (disconnection request)

t_disrq() closes down the specified connection, or rejects the connection indication of a calling TS application. In both cases the remote TS application receives a disconnect indication with the reason T_USER.

Either partner may close down the connection, regardless of which one actively set it up.

Along with the disconnection the remote TS application may be sent user data, if the transport system provides this option.

The t_disrq() call may overtake data that is still in transit. This data is then lost.

```
#include <cmx.h>
```

```
int t_disrq(tref, opt)
int *tref;
t_opt2 *opt;
```

-> tref

Pointer to a field containing the transport reference of the connection to be closed down.

-> opt

For opt, specify the value NULL or a pointer to a union containing a structure with system options. This union is used to specify the user data that the remote TS application is to receive along with the disconnection indication.

If opt = NULL is specified, CMX uses the default values specified.

The following structure is defined in <cmx.h>:

```
struct t_optc2 {
->  int   t_optnr;    /* Option no. */
->  char *t_udatap;   /* Data buffer */
->  int   t_udatal;   /* Length of the data buffer */
};
```

t_optnr

Option number. Specify T_OPTC2.

t_udatap

Pointer to a storage area containing user data to be received by the remote TS application.

Default value if NULL specified: Undefined

t_udatal

Length of the user data to be passed in the storage area t_udatap. If t_udatal = 0 is specified, t_udatap is ignored. The maximum value for t_udatal depends on the transport system (see the Release Notice).

Default value if NULL specified: 0

Return values

T_OK

The call was successful.

T_ERROR

Error. Query error code using t_error().

Errors

If an error occurs the following error values are possible. They can be queried by calling `t_error()`.

For error type `T_CMXTYPE` and error class `T_CMXCLASS`, the following may occur:

`T_EFAULT`

At least one of the pointers `opt` (`!= NULL`) or `t_udatap` (`!= NULL` and `t_ndatal != 0`) does not point to the process address space.

`T_WSEQUENCE`

The process is not attached for any TS application, or the connection specified in `tref` is neither set up nor being set up, nor is it being redirected.

`T_WPARAMETER`

The options specified in `opt` have an invalid format or contain illegal values.

`T_CCP_END`

The CCP is no longer operational.

In addition, the errors listed under `ioctl(2)` may occur.

See also

`t_detach()`, `t_disin()`, `t_event()`, `t_error()`

9.6.14 t_error - Error diagnosis (error)

t_error() returns diagnostic information when another CMX call returns T_ERROR.

The possible error messages for calls to the ICMX(L) program interface are generated either in the CMX library functions in the user process or in the operating system kernel. A distinction must be made between messages generated in CMX itself and those resulting from operating system calls in CMX.

The error messages generated by CMX are returned by t_error() in hexadecimal form. These error codes can be converted to readable text form with the aid of the calls t_strerror() and t_perror(). t_strerror() returns a pointer to a static area that contains the readable text form of an error message.

t_perror() writes the readable text form of an error message to stderr.

The hexadecimal error code can be decoded using the cmxdec command.

The format of CMX error messages is described in section 4.2.

```
#include <cmx.h>
```

```
int t_error()
```

Return values

The value returned by t_error() is the hexadecimal code for the error value generated by CMX. The error values are defined in <cmx.h>. A list of all possible error values of error type T_CMXTYPE (0) and error class T_CMXCLASS (0), i.e. all possible return values for t_error(), is provided in the appendix.

In the descriptions of the individual ICMX(L) function calls, the error values that t_error() returns if a particular function terminates in error are listed under the heading "Errors".

Files

<cmx.h> # Global CMX definition file

<tnsx.h> # TNSX definition file

<errno.h> # Messages for system calls

See also

t_perror(), t_strerror

9.6.15 t_event - Await or query event (event)

t_event() determines whether a CMX event has arrived for the current process.

The parameter cmode specifies the processing mode of t_event(). t_event() can:

- synchronously wait for a CMX event for the current process to arrive. While waiting, the process is suspended.
Waiting can be interrupted using signals.
A time limit for synchronous waiting may be specified in the opt options. If no event arrives within this waiting period, waiting is terminated.
- asynchronously check whether a CMX event for the current process has arrived. The function always returns immediately to the current process.

Along with the appropriate event, t_event() returns:

- the transport reference of the connection involved, to permit the event to be associated with the appropriate connection (tref parameter),
- event-specific additional information, if this has been specified in the opt options.

In addition, t_event() permits CMX to signal the arrival of more data for a connection, if data indications for the connection have not been explicitly blocked via t_datastop() or t_xdatstop(). If a T_DATAIN or T_XDATIN event is indicated for a process, t_event() may not be called again until the current process has accepted the indicated data with t_datain() or t_xdatin().

If several events are present for a connection, they are indicated one after another in the order in which they arrived.

Exceptions:

- A T_XDATIN event (expedited data received) may overtake T_DATAIN events (normal data received) without destroying them.
- A T_DISIN event (disconnection indication) may overtake T_DATAIN and T_XDATIN events for the connection involved and thus destroy them. The data that T_DATAIN/T_XDATIN was to have indicated is lost.

```
#include <cmx.h>
```

```
int t_event(tref, cmode, opt)
int *tref;
int cmode;
t_opt *opt;
```

<- tref

Pointer to a field in which CMX returns the connection-specific transport reference. The transport reference specifies the connection to which the event belongs. For the events T_NOEVENT and T_ERROR the contents of tref are undefined.

-> cmode

cmode is used to specify whether t_event() is to synchronously wait for an event or is to asynchronously check whether an event has arrived.

Possible values:

T_WAIT (synchronous processing)

The current process is suspended until a TS event arrives, the specified waiting time elapses (`t_timeout` parameter in opt) or a signal occurs (e.g. `alarm(CES)`). In the last two cases the event `T_NOEVENT` is returned.

T_CHECK (asynchronous processing)

The current process checks whether a TS event is waiting.

If a TS event is waiting for the current process, the event is returned to the process.

If no event is waiting, the event `T_NOEVENT` is returned to the process.

-> opt

For opt, you may specify `NULL` or a pointer to a union containing structures with system options.

If `NULL` is specified, CMX uses the defined default values.

The following structure is defined in `<cmx.h>`:

```

    struct t_opte1 {
->     int  t_optnr;      /* Option no. */
<-     int  t_attid;     /* CMX attachment reference */
<-     int  t_uattid;    /* User attachment reference */
<-     int  t_ucepid;    /* User connection reference */
->     int  t_timeout;   /* Time limit for T_WAIT */
<-     int  t_evdat;     /* Event-specific information */
    };

    struct t_opte2 {
->     int  t_optnr;      /* Option no. */
<-     int  t_attid;     /* CMX attachment reference */
<-     int  t_uattid;    /* User attachment reference */
<-     int  t_ucepid;    /* User connection reference */
->     int  t_timeout;   /* Time limit for T_WAIT */
<-     int  t_evdat;     /* Event-specific information */
<-     int  t_evinfo[10]; /* BS2000 event information */
    };

```

t_optnr

Option number. Specify

`T_OPTE1` in `t_opte1`

`T_OPTE2` in `t_opte2`

t_attid

In `t_attid t_event()` returns the CMX-internal reference for the attachment involved.

The CMX reference is also returned by CMX as an option in `t_attach()`. It serves only trace and diagnostic purposes and is used exclusively for logging.

t_uattid

In `t_uattid t_event()` returns the user reference for the attachment involved.

The user reference is passed to CMX as an option in `t_attach`. This enables a process that controls multiple TS applications to associate a TS event with the appropriate attachment of a TS application.

t_ucepid

In `t_ucepid t_event()` returns the user reference for the connection involved for the TS events `T_CONCF`, `T_DATAIN`, `T_XDATIN`, `T_DATAGO`, `T_XDATGO` and `T_DISIN`.

The user reference is passed to CMX in `t_conrq()` or `t_conrs()`. This enables a process that maintains multiple connections to associate a TS event with the appropriate connection. This feature, selected by the user, constitutes an alternative to the transport reference `tref`, defined by CMX.

t_timeout

With cmode = T_WAIT:

For t_timeout a waiting period may be specified during which t_event() is to synchronously wait for an event.

With cmode = T_CHECK:

Any value specified for t_timeout is ignored.

Possible specifications for t_timeout:

T_NOLIMIT

No waiting period is defined. The process waits (without time limit) until an event arrives or t_event() is terminated by a signal.

T_NO

The process does not wait. It resumes immediately with any TS event present or with T_NOEVENT (corresponds to cmode = T_CHECK).

n > 0

The process waits n seconds for the arrival of a TS event. If no TS event for the waiting process arrives within this time period, the process resumes with the event T_NOEVENT. Waiting may be terminated by means of signals.

Default value if NULL specified: T_NOLIMIT

t_evdat

Here, CMX returns event-specific additional information.

Possible information:

With the events T_DATAIN and T_XDATIN the length of the indicated data is specified here.

With the other TS events, including T_NOEVENT, the additional information is undefined.

t_evinf[10]

This field is used by BS2000 applications and is not supported by CMX in SINIX.

Return values**T_CONIN**

This event indicates that a calling TS application wishes to set up a connection to the current process. This connection indication must first be fetched with t_conin(), then confirmed with t_conrs() or rejected with t_disrq().

T_CONCF

This event indicates that the called TS application has responded positively to a connection request of the current process.

This connection setup confirmation must be fetched with t_concf().

T_DATAIN

This event indicates that data has been received via the connection specified in tref. The data must be fetched with t_datain(). CMX does not indicate this event for a connection so long as data flow on it is blocked, i.e. when the receiving process has issued t_datastop() for it.

T_DATAGO

The local TS application may resume sending data on the connection specified in tref.

Possible reaction: t_datarq().

The event T_DATAGO also permits the local TS application to resume sending expedited data on this connection, assuming the sending and receiving of expedited data was agreed at connection setup.

T_DISIN

This event indicates disconnection of the connection specified in `tref`.
This disconnect indication must be fetched with `t_disin()`.

T_ERROR

Error. Query error code using `t_error()`.

T_NOEVENT

This event means:

If `cmode = T_CHECK`

No event waiting.

If `cmode = T_WAIT`

Wait status of the process terminated, either by signal or because the specified waiting period elapsed. No TS event arrived.

The contents of `tref` are undefined.

T_XDATIN

This event indicates that expedited data has been received on the connection specified in `tref`. The data must be fetched with `t_xdatin()`.

This event is indicated only:

- if the exchange of expedited data was agreed at connection setup, and
- while the flow of expedited data on the connection is not blocked. The flow of expedited data is blocked when the receiving process has issued `t_xdatstop()` for the connection.

T_XDATGO

With this event CMX indicates that the process may resume sending expedited data on the connection specified in `tref`.

Possible reaction: `t_xdatrq()`.

CMX indicates this event only if the exchange of expedited data was agreed at connection setup.

Errors

If an error occurs the following error values are possible. They can be queried by calling `t_error()`.

For error type `T_CMXTYPE` and error class `T_CMXCLASS`, the following may occur:

T_EFAULT

The pointer `opt` (`!= NULL`) does not point to the process address space.

T_WSEQUENCE

The process is not attached for any TS application, or a TIDU or ETSDU has not yet been completely read.

T_WPARAMETER

The value specified in `cmode` is invalid, or the options specified in `opt` have an invalid format or contain illegal values.

T_CBRECURSIVE

Recursive `t_event` call in a callback routine is not permitted.

In addition, the errors listed under `ioctl(2)` may occur.

See also

t_attach(), t_callback(), t_concf(), t_conin(), t_datain(), t_datago(), t_datastop(), t_disin(), t_error(),
t_xdatin(), t_xdatgo(), t_xdatstop()

9.6.16 t_getaddr - Query TRANSPORT ADDRESS (get address)

t_getaddr() ascertains the TRANSPORT ADDRESS of a remote TS application from TS directory 1.

For the parameter globname, specify the GLOBAL NAME of the TS application.

t_getaddr() returns a pointer to a static area containing the TRANSPORT ADDRESS of the TS application.

This static area is overwritten at each call. If the contents of the area must be saved, the caller must copy the area.

The length of the area to be copied is obtained from the length field t_palng defined in struct t_partaddr.

```
#include <cmx.h>
```

```
struct t_partaddr *t_getaddr(globname, opt)
char *globname;
t_optg *opt;
```

-> globname

For this parameter, specify the GLOBAL NAME of the TS application whose TRANSPORT ADDRESS you wish to obtain.

The GLOBAL NAME is to be specified as a NULL-terminated string in the form

"NP5.NP4.NP3.NP2.NP1"

The items NP_i (i=1,2,3,4,5) represent the name parts of the hierarchically-structured GLOBAL NAME. NP5 is name part[5], i.e. the name part at the lowest hierarchical level. NP1 is name part[1], i.e. the highest name part in the hierarchy. The remaining name parts must be specified in increasing hierarchical order from left to right.

If one of the name parts for a particular GLOBAL NAME has no value (e.g. NP4), and this name part is followed by another name part that is higher in the hierarchy (e.g. NP3), the separator (.) from the name part with no value must be specified.

A series of separators appearing at the end of the value of globname may be omitted.

The GLOBAL NAME is then specified as follows: "NP5..NP3"

At least one of the name parts NP_i must be specified.

If the separator character (.) is a component of a name part, it must be represented as \. (backslash period).

Examples:

1. GLOBAL NAME:

Name part[1] = D
 Name part[2] = SIEMENS-AG
 Name part[3] = MCH-P
 Name part[4] = DF1
 Name part[5] = G.MEIER

Specification for globname:

"G\MEIER.DF1.MCH-P.SIEMENS-AG.D"

2. GLOBAL NAME:

Name part[2] = BU&B
 Name part[5] = PENCILPUSHER

Specification for globname:

"PENCILPUSHER...BU&B"

-> opt

Pointer to a unit with system options or NULL.

CMX (UNIX) ignores this parameter. The following structure is only defined in <cmx.h> to maintain compatibility with CMX (BS2000).

```
struct t_optg1 {
->  int  t_optnr;      /* Option no. */
->  int  t_evref;      /* System event reference point */
->  char *t_buf[200]; /* Work area */
};
```

t_optnr

Option number. Specify:

T_OPTG1 in t_optg1

t_evref

This field is used by BS2000 and is not supported by CMX (UNIX).

t_buf[200]

This field is used by BS2000 and is not supported by CMX (UNIX).

Return values

If the call was successful, t_getaddr() returns a pointer to a storage area containing the TRANSPORT ADDRESS.

In case of error, t_getaddr() returns a NULL pointer. The error code can be queried using t_error().

Errors

If an error occurs the following error values are possible. They can be queried by calling t_error().

Error values of error types T_DSTEMP_ERR, T_DSCALL_ERR, T_DSPERM_ERR and T_DSWARNING may occur.

For error class T_DSPAR_ERR, the following error values may occur:

T_DIRERR

TS directory DIR1 is not present.

T_NAMERR

The GLOBAL NAME specified in globname does not exist in TS directory 1.

T_ILLNAM

The GLOBAL NAME specified in globname is syntactically invalid (too many name parts, invalid lengths of name parts, invalid characters within the name).

T_PROPER

There is no TRANSPORT ADDRESS assigned to the GLOBAL NAME specified in globname.

Possible error values with error class T_DSSYS_ERR are the system error messages defined in <errno.h>.

For error class T_DSILL_VERS, the following error value may occur:

T_NOTSPEC

The CMX version linked into the process and the CMX runtime environment are incompatible.

For error class T_DSINT_ERR, the following error values are possible:

T_TIMEOUT

The TNSX daemon tnsxd does not respond within 20 seconds.

T_PROT

Errors occurred in the protocol with tnsxd.

T_LFILE

TS directory 1 (DIR1) has an incorrect format.

See also

t_error()

9.6.17 t_getfds - get file descriptors in use

`t_getfds()` returns the file descriptors currently in use by CMX connections. These file descriptors may be used as an input to the UNIX function call `poll()`. With this function call the user can wait not only for CMX specific events. However, one can also wait for other event classes (e.g. user input).

If the `poll()` system call recognizes events on file descriptors, `t_event()` must then be called. Please note that the `t_event()` call can return `T_NOEVENT`.

The file descriptor assigned to an CMX connection is only known after the CMX connection has been established. This means that `poll()` can only be used to receive events on CMX connections that are already established at the time of the `t_getfds()` call (in other words the file descriptor is known).

The file descriptors used by the CMX programming interface must be checked before every call to `poll()` since they change dynamically.

```
#include <cmx.h>
```

```
int t_getfds (struct pollfd **poll_fds, int *nfdes)
```

<- poll_fds

In `poll_fds`, the caller transfers a pointer containing a pointer to a structure `poll_fd`.

After successful execution, the caller receives a pointer to an array of structures of the type `pollfd`. This array contains all the file descriptors currently being used by CMX. The array contains an element of the `pollfd` structure for every file descriptor used by CMX. The number of elements of this array is returned in `nfdes`.

The `pollfd` structure contains the following elements:

```
int fd;                /* file descriptor */
short events;          /* requested events */
short revents;         /* returned events */
```

For further information, refer to the UNIX Programmer's Reference Manual description of the UNIX system call `poll()`.

If you do not want the user program to wait for additional events, the return values of `t_getfds()` can be transferred directly to the `poll()` system call.

<- nfdes

Pointer to an integer variable.

After successful execution, this integer variable contains the number of elements in the `poll_fds` array. The number of elements in the `poll_fds` array corresponds to the number of file descriptors currently being used by CMX.

Return values

If the call was successful, `t_getfds()` returns the following values:

- 0 Call is ok: the file descriptors have not changed since the last call.
- 1 Call is ok: the file descriptors have changed since the last call.

Errors

If an error occurs, `t_getfds()` returns the following values.

-1 Error occurred

Example of `t_getfds()`

```
struct pollfd    *fdes;
int              nfdes;
int              timeout = 5000; /*5 seconds */
int              ret;

...

if (t_getfds (&fdes, &nfdes) >= 0)
{
    /* preparation before calling poll */

    ...

    ret = poll(fdes, nfdes, timeout);
    if (ret > 0)
    {
        /* event received */
        /* call t_event */

        ...
    }
    else if (ret == 0)
    {
        /* timeout when calling poll */

        ...
    }
    else
    {
        /* poll returned an error */

        ...
    }
}

...
```

For an exact description of the `poll()` system call, refer to the UNIX programmer's reference manual.

In the example above, additional file descriptions may be contained in the `fdes` array after the `t_getfds()` call and before the `poll()` call with which, for example, entries made by a user on the user interface can be received.

With `t_getfds()` and `poll()` you can therefore implement a common waiting point within a UNIX process for all events that can be received in the particular UNIX process.

9.6.18 t_getloc - Query LOCAL NAME (get local name)

t_getloc() ascertains the LOCAL NAME of a TS application from TS directory 1.

For the parameter globname, specify the GLOBAL NAME of the TS application.

t_getloc() returns a pointer to a static area containing the LOCAL NAME of the TS application.

This static area is overwritten at each call. If the contents of the area must be saved, the caller must copy the area. The length of the area to be copied is obtained from the length field t_mnln in struct t_myname.

```
#include <cmx.h>

struct t_myname *t_getloc(globname, opt)
char *globname;
t_optg *opt;
```

-> globname

For this parameter, specify the GLOBAL NAME of the TS application whose LOCAL NAME you wish to obtain.

The GLOBAL NAME is to be specified as a NULL-terminated string in the form "NP5.NP4.NP3.NP2.NP1"

The items NP_i (i=1,2,3,4,5) represent the name parts of the GLOBAL NAME. NP5 is name part[5], i.e. the name part at the lowest hierarchical level. NP1 is name part[1], i.e. the highest name part in the hierarchy. The remaining name parts must be specified in increasing hierarchical order from left to right.

If one of the name parts for a particular GLOBAL NAME has no value (e.g. NP4), and this name part is followed by another name part that is higher in the hierarchy (e.g. NP3), the separator (.) from the name part with no value must be specified. A series of separators appearing at the end of the value of globname may be omitted.

The GLOBAL NAME is then specified as follows: "NP5..NP3"

At least one of the name parts NP_i must be specified.

If the separator character (.) is a component of a name part, it must be represented as \. (backslash period).

Examples:

GLOBAL NAME:

```
Name part[1] = D
Name part[2] = SIEMENS-AG
Name part[3] = MCH-P
Name part[4] = DF1
Name part[5] = G.MEIER
```

Specification for globname:

```
"G\MEIER.DF1.MCH-P.SIEMENS-AG.D"
```

GLOBAL NAME:

```
Name part[2] = BU&B
Name part[5] = PENCILPUSHER
```

Specification for globname:

```
"PENCILPUSHER...BU&B"
```

-> opt

Pointer to a unit with system options or NULL.

CMX (UNIX) ignores this parameter. The following structure is only defined in <cmx.h> to maintain compatibility with CMX (BS2000).

```
struct t_optg1 {
->  int  t_optnr;    /* Option no. */
->  int  t_evref;    /* System event reference point */
->  char *t_buf[200]; /* Work area */
};
```

t_optnr

Option number. Specify:

T_OPTG1 in t_optg1

t_evref

This field is used by BS2000 and is not supported by CMX (UNIX).

t_buf[200]

This field is used by BS2000 and is not supported by CMX (UNIX).

Return values

If the call was successful, t_getloc() returns a pointer to a storage area containing the LOCAL NAME.

In case of error, t_getloc() returns a NULL pointer. The error code can be queried using t_error().

Errors

If an error occurs the following error values are possible. They can be queried by calling t_error().

Error values of error types T_DSTEMP_ERR, T_DSCALL_ERR, T_DSPERM_ERR and T_DSWARNING may occur.

For error class T_DSPAR_ERR, the following error values may occur:

T_DIRERR

TS directory 1 (DIR1) is not present.

T_NAMERR

The GLOBAL NAME specified in globname does not exist in TS directory 1.

T_ILLNAM

The GLOBAL NAME specified in globname is syntactically invalid (too many name parts, invalid lengths of name parts, invalid characters within the name).

T_PROPER

There is no LOCAL NAME assigned to the GLOBAL NAME specified in globname.

Possible error values with error class T_DSSYS_ERR are the system error messages defined in <errno.h>.

For error class T_DSILL_VERS, the following error value may occur:

T_NOTSPEC

The CMX version linked into the process and the CMX runtime environment are incompatible.

For error class T_DSINT_ERR, the following error values are possible:

T_TIMEOUT

The TNSX daemon tnsxd does not respond within 20 seconds.

T_PROT

Errors occurred in the protocol with tnsxd.

T_LFILE

TS directory 1 (DIR1) has an incorrect format.

See also

t_error()

9.6.19 t_getname - Query GLOBAL NAME (get name)

Given the TRANSPORT ADDRESS of a remote TS application, t_getname() ascertains its GLOBAL NAME from TS directory 1.

The TRANSPORT ADDRESS of the TS application must be specified by the caller in the parameter addr.

t_getname() returns a pointer to a static area containing the GLOBAL NAME of the TS application.

This static area is overwritten at each call. If the contents of the area must be saved, the caller must copy the area.

The GLOBAL NAME is returned by CMX as a NULL-terminated string in the form NP5.NP4.NP3.NP2.NP1

The items NP_i (i=1,2,3,4,5) represent the name parts of the GLOBAL NAME. NP5 is name part[5], i.e. the name part at the lowest hierarchical level. NP1 is name part[1], i.e. the highest name part in the hierarchy. The remaining name parts are specified in increasing hierarchical order from left to right.

If one of the name parts for a particular GLOBAL NAME has no value (e.g. NP4), and this name part is followed by another name part that is higher in the hierarchy (e.g. NP3), the separator (.) from the name part with no value is nevertheless returned.

A series of separators appearing at the end of the value of globname is omitted.

The GLOBAL NAME is then specified by CMX as follows: "NP5..NP3"

If the separator character (.) is a component of a name part, it is represented as \. (backslash period).

```
#include <cmx.h>
```

```
char *t_getname(addr, opt)
struct t_partaddr *addr;
t_optg *opt;
```

-> addr

Pointer to a storage area with the TRANSPORT ADDRESS

-> opt

Pointer to a unit with system options or NULL.

CMX (UNIX) ignores this parameter. The following structure is only defined in <cmx.h> to maintain compatibility with CMX (BS2000).

```
struct t_optg1 {
->  int  t_optnr;    /* Option no. */
->  int  t_evref;    /* System event reference point */
->  char *t_buf[200]; /* Work area */
};
```

t_optnr

Option number. Specify:

T_OPTG1 in t_optg1

t_evref

This field is used by BS2000 and is not supported by CMX (UNIX).

t_buf[200]

This field is used by BS2000 and is not supported by CMX (UNIX).

Return values

If the call was successful, t_getname() returns a pointer to a storage area containing the GLOBAL NAME.

In case of error, `t_getname()` returns a NULL pointer. The error code can be queried using `t_error()`.

Errors

If an error occurs the following error values are possible. They can be queried by calling `t_error()`. Error values of error types `T_DSTEMP_ERR`, `T_DSCALL_ERR`, `T_DSPERM_ERR` and `T_DSWARNING` may occur.

For error class `T_DSPAR_ERR`, the following error values may occur:

`T_DIRERR`

TS directory DIR1 is not present.

`T_LENERR`

The length field `t_palng`, contained in the TRANSPORT ADDRESS specified in `addr`, has an invalid value.

Possible error values with error class `T_DSSYS_ERR` are the system error messages defined in `<errno.h>`.

For error class `T_DSILL_VERS`, the following error value may occur:

`T_NOTSPEC`

The CMX version linked into the process and the CMX runtime environment are incompatible.

For error class `T_DSINT_ERR`, the following error values are possible:

`T_TIMEOUT`

The TNSX daemon `tnsxd` does not respond within 20 seconds.

`T_PROT`

Errors occurred in the protocol with `tnsxd`.

`T_LFILE`

TS directory 1 (DIR1) has an incorrect format.

For error class `T_DSMESSAGE`, the following error value is possible:

`T_LEAFNO`

In TS directory 1 there are either none or more than one GLOBAL NAME to which the TRANSPORT ADDRESS specified in `addr` is assigned.

See also

`t_error()`

9.6.20 t_info - Query information on CMX (information)

t_info() provides information on the TIDU length. Generally, this information only becomes available after the transport connection has been set up completely.

```
#include <cmx.h>
```

```
int t_info(tref, opt)
int *tref;
t_opti *opt;
```

-> tref

Pointer to a field with the transport reference of the connection.

<> opt

For opt, you may specify NULL or a pointer to a union containing structures with system options. The following structure is defined in <cmx.h>:

```
    struct t_opti1 {
->      int t_optnr;          /* Option no. */
<-      int t_maxl;          /* TIDU length */
    };

    struct t_opti2 {
->      int t_optnr;          /* Option no. */
<-      int t_evref;          /* System reference point */
<-      int t_buffer[180];    /* Buffer for Name Service
                                output */
    };
```

t_optnr

Option number. Specify:

T_OPTI1 in t_opti1

T_OPTI2 in t_opti2

t_maxl

In this field CMX enters the length of a TIDU.

This value specifies the maximum number of bytes that can be sent to CMX or received from CMX per call when transferring data over this connection.

t_evref

This field is used by BS2000 applications and is not supported by CMX (UNIX).

t_buffer[180]

This field is used by BS2000 applications and is not supported by CMX (UNIX).

Return values

T_OK

The call was successful.

T_ERROR

Error. Query error code using t_error().

Errors

If an error occurs the following error values are possible. They can be queried by calling `t_error()`.

For error type `T_CMXTYPE` and error class `T_CMXCLASS`, the following may occur:

`T_EFAULT`

The pointer `opt` (`!= NULL`) does not point to the process address space.

`T_WSEQUENCE`

The process is not attached for any TS application, or

the desired information is not available. `t_info()` can only be executed when the connection has been set up, because only then can information be output about the connection.

`T_WPARAMETER`

The options passed via `opt` have an invalid format or contain illegal values.

`T_CCP_END`

The CCP is no longer operational.

In addition, the errors listed under `ioctl(2)` may occur.

9.6.21 t_perror - Output CMX error message in decoded form

t_perror() decodes CMX error messages passed to the process in hexadecimal form by CMX when t_error() is called.

t_perror() writes the plain English form of the CMX error message specified in code to the standard error output stderr.

In the s parameter an additional explanatory text may be specified, e.g an indication of the CMX call and TS application to which the error refers.

Format of output from t_perror():

t_perror() first writes the text specified with s (if s != NULL), then : (colon) and \n (newline). This is followed by the plain English form of the CMX error message passed. This text consists of the error symbols, as defined in <cmx.h>, and an accompanying text. Each error symbol is preceded by \t. Each accompanying text ends with \n.

The accompanying text is taken from the message file cmxlib.cat. It will not be output if cmxlib.cat is not available on your system. The format of cmxlib.cat is dependent on the operating system and the set language variable. See the appropriate system manual for more details.

```
#include <cmx.h>
```

```
void t_perror(s, code)
```

```
char *s;
```

```
int code;
```

-> s

Pointer to a storage area containing text that is to precede the readable text form of the error message, or the value NULL.

-> code

For code, specify the representation of the error message that was passed to the process by CMX when t_error() was called.

Files

cmxlib.cat - Message file

See also

t_error(), t_strerror()

9.6.22 t_preason - Decode and output reasons for disconnection

t_preason() decodes reasons for disconnection passed to the process in hexadecimal form when t_disin() is called.

t_preason() writes the plain English form of the reason for disconnection specified in reason to the standard error output stderr.

In the s parameter an additional explanatory text may be specified, e.g an indication of the connection or TS application to which the output refers.

Format of output from t_preason():

t_preason() first writes the text specified with s (if s != NULL), then : (colon) and \n (newline). This is followed by the plain English form of the disconnection reason passed. This text consists of the symbol for the disconnection reason, as defined in <cmx.h>, and an accompanying text. The symbol for the disconnection reason is preceded by \t. The accompanying text ends with \n.

The accompanying text is taken from the message file cmxlib.cat. It will not be output if cmxlib.cat is not available on your system. The format of cmxlib.cat is dependent on the operating system and the set language variable. See the appropriate system manual for more details.

```
#include <cmx.h>
```

```
void t_preason(s, reason)
char *s;
int reason;
```

-> s

Pointer to a storage area containing text that is to precede the plain English form of the disconnection reason, or the value NULL.

-> reason

For reason, specify the representation of the disconnection reason that was passed to the process by CMX when t_disin() was called.

Files

cmxlib.cat - Message file

See also

t_disin(), t_strreason()

9.6.23 t_setopt - Set options in CMX (set options)

t_setopt can be used to switch options on and off in CMX.

In this version the option T_DEBUG only is provided for activating/deactivating library traces.

```
#include <cmx.h>
```

```
int t_setopt(component, opt)
int component;
t_opts *opt;
```

-> component

Specifies in which CMX component the option should be set.

Possible values:

T_LIB

The set option is a library option.

-> opt

Pointer to a union that contains an options structure.

The following structure is defined in <cmx.h>:

```
struct t_opts1 {
->  int t_optnr;          /* Option no.  */
->  int t_optname;        /* Option name */
->  char *t_optvalue;     /* Pointer to options string */
};
```

t_optnr

Option number. Specify T_OPTS1.

t_optname

Specifies the option that is to be switched on or off.

Possible values:

T_DEBUG

Activate/deactivate trace mechanism.

t_optvalue

Pointer to a (null-terminated) string that contains the option value. If the string is empty, the option specified in t_optname is switched off. The contents of the string depends on the value of t_optname.

t_optname = T_DEBUG: The format of the trace options is the same as for the environment variable CMXTRACE .

Return value

T_OK

The call was successful.

T_ERROR

Error. Query error code with t_error().

Errors

If an error occurs the following error values are possible. They can be queried by calling `t_error()`.

For error type `T_CMXTYPE` and error class `T_CMXCLASS`, the following may occur:

T_WPARAMETER

The value specified in component is invalid or the option specified in opt has an incorrect format or contains incorrect values.

9.6.24 t_strerror - Decode CMX error message

t_strerror() decodes CMX error messages passed to the process in hexadecimal form by CMX when t_error() is called.

t_strerror() returns a pointer to a static area that contains the plain English form of the CMX error message specified in code.

This text consists of error symbols, as defined in <cmx.h>, and accompanying text. Each error symbol is preceded by \t. Each accompanying text ends with \n.

The accompanying text is taken from the message file cmxlib.cat. It will not be output if cmxlib.cat is not available on your system. The format of cmxlib.cat is dependent on the operating system and the set language variable. See the appropriate system manual for more details.

```
#include <cmx.h>
```

```
char *t_strerror(code)  
int code;
```

-> code

For code, specify the representation of the error message that was passed to the process by CMX when t_error() was called.

Return values

If the call was successful, t_strerror() returns a pointer to a storage area with the plain English form of the CMX error message as a C string.

If an undefined value is specified in code, t_strerror() returns a pointer to the text:

```
"\t<code> Cannot decode\n"
```

In case of error, t_strerror() returns a NULL pointer.

Files

cmxlib.cat - Message file

See also

t_error(), t_perror()

9.6.25 t_strreason - Decode reasons for disconnection

t_strreason() decodes reasons for disconnection passed to the process in hexadecimal form when t_disin() is called.

t_strreason() returns a pointer to a static area that contains the plain English form of the reason for disconnection specified in reason.

This text consists of the symbol for the disconnection reason, as defined in <cmx.h>, and an accompanying text. The symbol for the disconnection reason is preceded by \t. The accompanying text ends with \n.

The accompanying text is taken from the message file cmxlib.cat. It will not be output if cmxlib.cat is not available on your system. The format of cmxlib.cat is dependent on the operating system and the set language variable. See the appropriate system manual for more details.

```
#include <cmx.h>
```

```
char *t_strreason(reason)
int reason;
```

-> reason

For reason, specify the representation of the disconnection reason that was passed to the process by CMX when t_disin() was called.

Return values

If the call was successful, t_strreason() returns a pointer to a storage area with the plain English form of the disconnection reason as a C string.

If a value is specified in reason that is not defined, t_strreason() returns a pointer to the text:

```
"\t<reason> Cannot decode\n"
```

In case of error, t_strreason() returns a NULL pointer.

Files

cmxlib.cat - Message file

See also

t_disin(), t_preason()

9.6.26 t_xdatgo - Release the flow of expedited data (expedited data go)

t_xdatgo() releases the blocked flow of expedited data on the specified connection. By means of this call the current process informs CMX that it is again ready to receive expedited data.

More specifically, the call has the following effects:

- The current process can again receive the event T_XDATIN for the specified connection, if one is waiting.
- The sending TS application receives the event T_XDATGO. It may again send data. Normal data is not affected by t_xdatgo().

t_xdatgo() may be called only if the exchange of expedited data was agreed when the connection was set up.

```
#include <cmx.h>
```

```
int t_xdatgo(tref)
int *tref;
```

-> tref

Pointer to a field with the transport reference of the connection on which the flow of expedited data is to be released again.

Return values

T_OK

The call was successful.

T_ERROR

Error. Query error code using t_error().

Errors

If an error occurs the following error values are possible. They can be queried by calling t_error().

For error type T_CMXTYPE and error class T_CMXCLASS, the following may occur:

T_WSEQUENCE

The process is not attached for any TS application, or

the process is not in the data phase for the connection specified in tref, or the exchange of expedited data was not agreed for this connection.

T_CCP_END

The CCP is no longer operational.

In addition, the errors listed under ioctl(2) may occur.

See also

t_event(), t_error(), t_xdatstop()

9.6.27 t_xdatin - Receive expedited data (expedited data indication)

t_xdatin() accepts a T_XDATIN event previously reported via t_event(). The t_xdatin() call must be made before the next t_event().

By means of this call the current process receives an Expedited Transport Service Data Unit (ETSDU) from the sending TS application on the specified connection. The maximum length of an ETSDU depends on the transport system used. However, it is never greater than T_EXP_SIZE bytes.

If the expedited data fits into the storage area datap provided, the value T_OK is returned. Otherwise, a value n > 0 is returned, where n is the number of bytes of the ETSDU that have not yet been read (remaining length). In this case, t_xdatin() must be called repeatedly until the entire ETSDU has been read. Only then can other CMX calls be issued again, e.g. t_event().

```
#include <cmx.h>
```

```
int t_xdatin(tref, datap, datal)
int *tref;
char *datap;
int *datal;
```

-> tref

Pointer to a field containing the transport reference of the connection, obtained via t_event().

<- datap

Pointer to a storage area in which CMX enters the data of the ETSDU received.

<> datal

Pointer to a field in which prior to the call the length of the data area datap must be entered. A value of at least 1 must be specified.

Following the call, CMX returns in this field the number of bytes entered.

Return values

T_OK

The call was successful. The expedited data was completely read.

n > 0

n bytes remain from the ETSDU.

T_ERROR

Error. Query error code using t_error().

Errors

If an error occurs the following error values are possible. They can be queried by calling t_error().

For error type T_CMXTYPE and error class T_CMXCLASS, the following may occur:

T_EFAULT

The pointer datap does not point to the process address space.

T_WSEQUENCE

The process is not attached for any TS application, or

the exchange of expedited data was not agreed for the connection specified in tref, or no T_XDATIN was indicated for the connection specified in tref.

T_WPARAMETER

The length specified in data1 is invalid.

T_COLLISION

The event T_DISIN (disconnect indication) occurred for the connection, but has not yet been queried with `t_event()`.

Action: call `t_event()`.

T_CCP_END

The CCP is no longer operational.

In addition, the errors listed under `ioctl(2)` may occur.

See also

`t_error()`, `t_event()`

9.6.28 t_xdatrq - Send expedited data (expedited data request)

t_xdatrq() sends an Expedited Transport Service Data Unit (ETSDU) with expedited data to the receiving TS application via the connection specified. The maximum length of a ETSDU depends on the transport system used. However, it is never greater than T_EXP_SIZE bytes (does not apply to WAN-X25).

The t_xdatrq() call is permitted only when the exchange of expedited data was agreed when the relevant connection was set up.

ETSDUs may overtake Transport Interface Data Units (TIDUs) with normal data that had been sent earlier. It is guaranteed that ETSDUs will never arrive at the receiving TS application later than TIDUs sent after them.

If T_XDATSTOP is returned, the ETSDU has been accepted but the send flow of ETSDUs and TIDUs on this connection has been blocked.

The flow of expedited data can be blocked by:

- the receiving TS application, which can block the flow of ETSDUs by calling t_xdatstop(), or
- CMX, if the local buffer is full.

If the flow of ETSDUs is blocked, before further ETSDUs can be sent you must wait, by means of t_event(), for the event T_XDATGO or T_DATAGO for the connection.

Successful execution of t_xdatrq() (T_OK) does not mean that the receiving TS application has already accepted the data.

If t_xdatrq() fails (T_ERROR), this always indicates that a local error has been found.

```
#include <cmx.h>
```

```
int t_xdatrq(tref, datap, data)
int *tref;
char *datap;
int *data;
```

-> tref

Pointer to a field with the transport reference of the connection on which the expedited data is to be sent.

-> datap

Pointer to a storage area containing the ETSDU to be sent.

-> data

Pointer to a field containing the number of bytes to be sent from the storage area datap.

Minimum value: 1

Maximum value: T_EXP_SIZE

(T_EXP_SIZE is defined in <cmx.h>.)

Return values

T_OK

The call was successful; further expedited data may be sent immediately.

T_XDATSTOP

The call was successful, but further ETSDUs may not be sent until the event T_XDATGO or T_DATAGO has arrived for this connection.

T_ERROR

Error. Query error code using `t_error()`.

Errors

If an error occurs the following error values are possible. They can be queried by calling `t_error()`.

For error type `T_CMXTYPE` and error class `T_CMXCLASS`, the following may occur:

T_EFAULT

The pointer `datap` does not point to the process address space.

T_WSEQUENCE

The process is not attached for any TS application, or

the process is not in the data phase for the connection specified in `tref`, or the exchange of expedited data was not agreed for the connection specified in `tref`, or the flow of expedited data is blocked for the connection specified in `tref`.

T_WPARAMETER

The length specified in `data1` is not permitted.

T_COLLISION

The event `T_DISIN` (disconnect indication) has arrived for the connection, but has not yet been fetched with `t_event()`.

Response: Call `t_event()`.

T_CCP_END

The CCP is no longer operational.

In addition, the errors listed under `ioctl(2)` may occur.

See also

`t_error()`, `t_event()`, `t_xdatstop()`

9.6.29 t_xdatstop - Block the flow of expedited data (expedited data stop)

t_xdatstop() blocks the flow of both expedited and normal data on the specified connection.

More specifically, the effects of t_xdatstop() are:

- The current process tells CMX that, until further notice, it is not ready to receive normal or expedited data for this connection. However, a T_DATAIN event or a T_XDATIN event that has already been indicated must be responded to first.
- The current process no longer receives the events T_DATAIN and T_XDATIN for the specified connection. However, while the data flow is blocked it may call other CMX functions, e.g. to set up, close down or redirect an additional connection.
- The sending TS application receives the return value T_XDATSTOP when it calls t_xdatrq() and the return value T_DATASTOP when it calls t_datarq(). It may not send any more normal or expedited data.

The flow of expedited data is released with t_xdatgo() or with t_datago().

t_xdatstop() may be called only if the exchange of expedited data was agreed when the connection was set up.

```
#include <cmx.h>
```

```
int t_xdatstop(tref)
int *tref;
```

-> tref

Pointer to a field with the transport reference of the connection.

Return values

T_OK

The call was successful.

T_ERROR

Error. Query error code using t_error().

Errors

If an error occurs the following error values are possible. They can be queried by calling t_error().

For error type T_CMXTYPE and error class T_CMXCLASS, the following may occur:

T_WSEQUENCE

The process is not attached for any TS application, or the process is not in the data phase for the connection specified in tref, or a TIDU or an ETSDU has not yet been completely read on the specified connection, or the exchange of expedited data was not agreed for this connection.

T_CCP_END

The CCP is no longer operational.

In addition, the errors listed under ioctl(2) may occur.

See also

t_datago(), t_error(), t_event(), t_xdatgo(), t_xdatrq()

10 Configuring CMX Connections

10.1 Transport Name Service TNSX

The Transport Name Service TNSX is used by CMX applications for managing addressing information.

TNSX reads the address information from a directory, the TS directory (Transport System Directory). This TS directory contains the address information referenced by a symbolic name known as the global name. The TS directory must contain information about all CMX applications on the local system and about the communication partners on remote systems.

CMX provides several functions to access the transport name service TNSX:

<code>t_getloc</code>	Retrieves for a given global name the corresponding local name of a CMX application. This local name is used in attaching to CMX (<code>t_attach</code>).
<code>t_getaddr</code>	Retrieves for a given global name the corresponding transport address of a remote TS application. This transport address is used in establishing a transport connection (<code>t_conrq</code>).
<code>t_getname</code>	Retrieves for a given transport address of a remote TS application the corresponding global name. The transport address may be obtained by interpreting an incoming connection request (<code>t_conin</code>) of a remote TS application.

10.2 TNSX Daemon tnsxd

The TNSX daemon `tnsxd` is the central server process for all enquiries to the transport name service TNSX. A CMX application accesses the TS directory via the TNSX daemon to obtain the address information.

The TNSX daemon `tnsxd` is started when the operating system is started. It can also be started at any time with the following command:

```
/opt/etc/tnsxd &
```

10.3 TNSX Compiler tnsxcom

Entries in the TS directory are created using the TNSX compiler `tnsxcom` with which they can also be read and modified. The TNS entries made in the TS directory are transferred in a file when the TNSX compiler is called. The TNSX compiler expects the TNS entries in this file in a fixed format. Once the format has been checked, the TNSX compiler compiles the entries into the format of the TS directory and writes the entries to the TS directory.

The TNSX compiler **tnsxcom** is called as follows:

```
/opt/bin/tnsxcom [-D] [-u] [filename]
```

- D** Dump mode: `tnsxcom` outputs all the TNS entries in the TS directory to the "filename" file, This option excludes the "-u" option.
- u** Update mode: `tnsxcom` reads the TNS entries from the "filename" file, runs a syntax check and writes syntactically correct entries to the TS directory. Any existing entries are updated. This option excludes the "-D" option.

Calling `tnsxcom` with the "-u" option is only possible under the root ID.

filename Name of the file to which the TNS entries will be written (Option “-D”) or from which the TNS entries will be read (Option “-u”).

10.4 TNS Entries

The TNS entries transferred to the TNS compiler tnsxcom in the input file must have the syntax described below.

The protocol architectures (transport systems) supported by CMX are used depending on the address format in the TNS entries. The following transport systems are supported by CMX as it is part of SOFTNET S7/UNIX:

- OSI transport protocol with inactive network layer (null Internet).

When selecting the OSI transport protocol the use of more than one ethernet board in CMX applications is supported.

- RFC 1006 via TCP/IP

 **CMX enables the creation of TS applications connecting to a variety of end systems. Please note that these end systems may expect certain values for some addressing parameters or have restrictions on addressing parameters which can not be described here. Please refer for further details to the corresponding documentation of your end system.**

10.4.1 TNS Configuration for the OSI Transport Protocol

If a TNS entry has the following format, the OSI transport protocol with inactive network layer (null Internet) is selected as the transport system for a SOFTNET S7/UNIX connection. More than one ethernet boards are supported.

10.4.1.1 Local Name of a CMX Application

```
<global_name>\n      TSEL LANSBKA <local_tsel>
```

<global_name>\

The entry is made in the TS directory with this global name.

Character string with maximum 30 ASCII characters.

The name must be terminated with the “\” character.

TSEL

This keyword is fixed. The entries following the keyword in this line contain information regarding the local name of a CMX application.

LANSBKA

This keyword is fixed. It indicates the use of the OSI transport protocol with inactive network layer (null Internet) as the transport system.

<local_tsel>

Indicates the local transport selector (local TSEL). The local CMX application is addressable on the network using this transport selector. If active connection establishment to a remote TS application is selected, this transport selector is used as the calling TSEL.

The following formats are possible for <local_tsel>:

A'<string>' <string> character string with a maximum of 8 ASCII characters

X'<string>' <string> character string with a maximum of 8 hexadecimal digits

10.4.1.2 Transport Address of a remote TS Application**<global_name>**

TA LANSBKA <subnet_id> <ethernet_addr> <remote_tsel>

<global_name>

The entry is made in the TS directory with this global name.

Character string with maximum 30 ASCII characters.

The name must be terminated with the “\” character.

TA

This keyword is fixed. The entries following it in this line contain information regarding the transport address of a remote TS application.

LANSBKA

This keyword is fixed. This indicates the use of the OSI transport protocol with inactive network layer (null Internet) as the transport system.

<subnet_id>

If the OSI transport protocol is used, the CMX version as part of SOFTNET S7/UNIX supports operation with more than one Ethernet board. This parameter is used to address the Ethernet board used in active connection establishment to the remote TS application.

 **The <subnet_id> parameter depends on the particular operating system. Please refer to the SOFTNET S7/UNIX product information for the operating system you are using to check the correct value.**

The following format is mandatory for <subnet_id>:

X'<string>' <string> character string with exactly 4 hexadecimal digits.

<ethernet_addr>

This is the Ethernet address via which the remote TS application is addressable on the network.

The following format is mandatory for <ethernet_addr>:

Character string with exactly 12 hexadecimal characters without separators.

<remote_tsel>

This is the remote transport selector (also known as the “called TSEL”). The remote TS application is addressed on the network using this transport selector.

The following formats are possible for <remote_tsel>:

A'<string>' <string> character string with a maximum of 8 ASCII characters

X'<string>' <string> character string with a maximum of 8 hexadecimal octets

10.4.1.3 Example of the OSI Transport Protocol

Here are some examples for TNS entries for the OSI transport protocol:

```
local_1\
      TSEL  LANSBKA X'010F'
```

local_1

The example above shows a TNS entry stored under the global name “local_1” in the TS directory. The name is terminated with the “\” character.

TSEL LANSBKA X'010F'

This part contains the local name of a CMX application.

In the example above, the transport selector X'010F' is used to logon at the local transport system. The parameter “X” stands for hexadecimal, in other words the transport selector “010F” must be specified in hexadecimal format. Using this transport selector, the local CMX application can be obtained on the network. The “LANSBKA” parameter indicates the use of the OSI transport layer with inactive network layer (null Internet).

```
remote_1\
      TA    LANSBKA X'<subnet_id>' 080014151588 A'Softnet1'
```

remote_1

The example above shows a TNS entry stored under the global name “remote_1” in the TS directory. The name is terminated with the “\” character.

TA LANSBKA X'<subnet_id>' 080014151588 A'Softnet1'

This part contains the transport address of a remote TS application.

If a connection is established to the remote TS application, the Ethernet address 080014151588 and the transport selector A'Softnet1' are used. The parameter "A" stands for ASCII, in other words the "Softnet1" transport selector is interpreted as an ASCII string. The "LANSBKA" parameter indicates the use of the OSI transport layer with inactive network layer (null Internet).

X'<subnet_id>'

If the OSI transport layer is used, CMX as part of SOFTNET S7/UNIX supports operation with more than one Ethernet board. The parameter X'<subnet_id>' specifies the Ethernet board that is used for active establishment of the connection to the remote TS application. In other words, the particular Ethernet board is addressed using the <subnet_id> parameter. Since this parameter depends on the operating system being used, it is not possible to indicate a generally valid value for all operating systems supported by SOFTNET S7/UNIX in the above example. Please refer to the SOFTNET S7/UNIX product information for the operating system you are using to check the correct value.

10.4.2 TNS Configuration for RFC1006 via TCP/IP

The format described below for a TNS entry selects RFC1006 via TCP/IP as the transport system for a SOFTNET S7/UNIX connection.



The tcp port 102 is reserved for applications implementing the RFC1006. It is used by RFC1006 applications when waiting for incoming connection requests.

On UNIX systems this port is a privileged port. This port may only be used by applications which have access rights as superuser.

This implies for CMX applications which are working with RFC1006 over TCP/IP and waiting for incoming connections requests (parameter t_apmode==T_PASSIVE when calling t_attach()) that they also must run with access rights as superuser.



The tcp port 102 is used in CMX applications over RFC1006. A port number, which is used to receive tcp events, may be attached to only one application at a time. This implies for CMX application when using RFC1006 that the CMX function t_attach() with parameter t_apmode==T_PASSIVE may be called only once. If more than one CMX connection shall be used, the parameter t_conlim must be set to the appropriate value.

10.4.2.1 Local name of a CMX Application

```
<global_name>\
```

```
TSEL RFC1006 <local tsel>
```

<global_name>

The entry is made in the TS directory using this global name.

Character string with a maximum of 30 ASCII characters.

The name must be terminated with the “\” character.

TSEL

This keyword is fixed. The entries following this keyword in the line contain information regarding the local name of a CMX application.

RFC1006

This keyword is fixed. It indicates use of RFC1006 via TCP/IP.

<local_tsel>

This is the local transport selector (local TSEL). The local CMX application can be obtained on the network using this transport selector. With active connection establishment to a TS application, this transport selector is used as the calling TSEL.

The following formats are possible for <local_tsel>:

A'<string>' <string> character string with a maximum of 8 ASCII characters

X'<string>' <string> character string with a maximum of 8 hexadecimal digits

10.4.2.2 Transport Address of a remote TS Application**<global_name>**

TA RFC1006 <ip-addr> <remote_tsel>

<global_name>

The entry is made in the TS directory using this global name.

Character string with a maximum of 30 ASCII characters.

The name must be terminated with the “\” character.

TA

This keyword is fixed. The subsequent entries in this line contain information regarding the transport address of a remote TS application.

RFC1006

This keyword is fixed. This indicates use of RFC1006 via TCP/IP.

<ip-addr>

This is the Internet address with which the remote TF application is addressable on the network.

The following format is fixed for <ip-addr>:

<number>.<number>.<number>.<number> where $0 \leq \text{number} < 256$

<remote tsel>

This is the remote transport selector (remote TSEL, also known as the called TSEL). The remote TS application is addressed on the network using this transport selector.

The following formats are possible for <remote tsel>:

A'<string>' <string> character string with a maximum of 8 ASCII characters

X'<string>' <string> character string with a maximum of 8 hexadecimal octets

10.4.2.3 Example of RFC1006 via TCP/IP

Here are some examples for TNS entries for RFC1006 via TCP/IP

```
local_2\
```

```
    TSEL RFC1006 A'1212'
```

```
local_2\
```

The example above shows a TNS entry stored under the global name "local_2" in the TS directory. The name is terminated with the "\ " character.

TSEL RFC1006 A'1212'

This part contains the local name of a CMX application.

In the example above, the transport selector A'1212' is used to log on at the local transport system. The parameter "A" stands for ASCII, in other words the transport selector "1212" must be specified in ASCII format and is interpreted as an ASCII string. The local CMX application can be obtained on the network using this transport selector. The "RFC1006" parameter indicates the use of RFC1006 via TCP/IP.

```
remote_2\
```

```
    TA RFC1006 192.33.21.23 A'4277'
```

```
remote_2\
```

The example above shows a TNS entry stored under the global name "remote_2" in the TS directory. The name is terminated with the "\ " character.

TA RFC1006 192.33.21.23 A'4277'

This part contains the transport address of a remote TS application.

When the connection is established to the remote TS application, the Internet address 192.33.21.23 and the transport selector A'4277' are used. The parameter "A" stands for ASCII, in other words the transport selector "4277" must be specified in ASCII format and is interpreted as an ASCII string. The "RFC1006" parameter indicates the use of RFC 1006 via TCP/IP.

10.4.3 Restrictions for Names in TNS Entries

The **logical name of a TNS entry** can be a maximum of 30 characters long. It must begin with a letter or the underscore character, the remaining 29 characters can be letters, underscores or numbers.

The **transport selectors** can be either in ASCII format preceded by an "A" or in hexadecimal format preceded by an "X".

When using the **ASCII format**, a maximum of 8 characters are possible for the **transport selector**. Letters, numbers or the underscore character can be used.

When using the **hexadecimal format** for the **transport selector**, an even number of a maximum 16 hexadecimal digits can be used. Hexadecimal digits include the numbers 0 to 9 and letters a to f or A to F.



Please note that there may be restrictions on the partner systems that go beyond the restrictions explained here.

10.5 Working with the TNSX Compiler

10.5.1 Generating TNS Entries

The TNS entries are generated using the TNSX compiler *tnsxcom*. The address information is edited in an ASCII file whose name is transferred to the TNS compiler as a parameter.

- Calling the TNS compiler to generate TNS entries:

/opt/bin/tnsxcom -u <ASCIIfile>

This call must be made as a super user.

The TNS entries in the specified file are included in the TS directory.

- Checking the newly generated entries:

/opt/bin/tnsxcom -D ASCIIfile

This call must be made as a super user.

All the TNS entries contained in the TS directory are written to the specified file.

10.5.2 Modifying Existing TNS Entries

If you want to change the address information of existing TNS entries, follow the procedure below:

- Write the current TNS entries to an ASCII file:

/opt/bin/tnsxcom -D ASCIIfile

This call does not need to be made as a super user.

All the TNS entries in the TS directory are written to the specified file.

- Change the address information in the ASCII file:

Edit the content of the ASCII file

- Update the TNS entries by calling the TNS compiler:

/opt/bin/tnsxcom -u ASCIIfile

This call must be made as a super user.

The TNS entries in the specified file are included in the TS directory or entries already in the TS directory are updated.

- Check the modifications made:

/opt/bin/tnsxcom -D ASCIIfile

This call does not need to be made as a super user.

The TNS entries currently in the TS directory are written to the specified file.

10.5.3 Deleting Existing TNS Entries

To delete an existing TNS entry, follow the procedure below:

- Generate an ASCII file with the following entry:

TNSname DEL

Where TNSname = the TNS entry to be deleted

- Call the TNS compiler:

/opt/bin/tnsxcom -u ASCIIfile

This call must be made as a super user.

All the TNS entries in the specified file are removed from the TS directory.

- Check that the entry has been deleted:

/opt/bin/tnsxcom -D ASCIIfile

This call does not need to be made as a super user

The entries remaining in the TS directory are written to the specified file.

10.5.4 Outputting the Address Information of all TNS Entries

The current address information of all existing TNS entries can be output using the following command. The output can be redirected to an ASCII file as follows:

/opt/bin/tnsxprop > ASCIIfile

Where ASCIIfile = any file name

All the entries currently in the TS directory are written to the file.

The format of the TNS entries output by `tnsxprop` is easier to read than the format output by calling `tnsxcom`.

11 CMX Library Trace

The trace of the CMX library is controlled by the environment variable CMXTRACE. By supplying the environment variable with a value, the trace is activated and the scope of the information to be collected is specified.

The trace entries of a process are collected as compact binary data in a dynamically created buffer and periodically written to temporary files. These binary files are edited separately with the *cmxl* tool. The binary files are saved in the */usr/tmp* directory. The file names consist of the prefix CMX and the process identification number *pid*.

cmxl reads the entries generated by the trace from the temporary file. The scope of the analysis is decided by the options selected for *cmxl*.

11.1 CMXTRACE: Controlling the Trace

The options specified in CMXTRACE control the trace. The options *s*, *S*, and *D* determine what is logged. The options *p*, *r* control the buffering and writing of the file:

CMXTRACE = "[-s] [-S] [-D] [-p fac] [-r wrap] [-f directory]"

- s** The CMX calls, their arguments, the options and user data are logged normally.
- S** The calls, their arguments, the contents of any options, the user data in their full length are logged.
 The options -s and -S exclude each other.
- D** The calls with additional information about system calls are logged in detail. This option is only available in addition to -s or -S.
- p fac** The decimal digit fac determines the buffer factor. The amount of buffering is determined according to $fac * BUFSIZ$ where BUFSIZ is determined by `<stdio.h>`.
 If you specify $fac = 0$, each trace entry is written to the file immediately (with no buffering).
 fac = 0..8
 Default: *fac* = 1
- r wrap** The decimal number wrap specifies that after
 $wrap * BUFSIZ$ bytes (BUFSIZ according to `<stdio.h>`)
 logging continues in the second temporary file `<directory>/CMXMa<pid>`. This second file handles the trace in exactly the same way as `CMXLa<pid>`. After $wrap * BUFSIZ$ bytes, the trace switches between the two files. Following this switch over the content of the file is overwritten.
 Default: *wrap* = 128
- f directory** The trace files are written to the specified directory.
 Default: *directory*: */usr/tmp* or */var/tmp*

11.2 cmxl: How the Trace is Processed

cmxl reads the entries generated by the trace from the temporary file, processes the entries according to the selected options and outputs the result to stdout.

The following options specify which trace entries from the file are processed. It is possible to specify more than one of the values described below per cmxl call. Only the options v and x exclude each other. If no options are specified, cdex is used as the default.

```
cmxl [-c] [-d] [-e] [-t] [-x] [-D] file
```

-c	The CMX calls for logging on and off the TS application with CMX and for establishing and terminating the connection are processed.
-d	The CMX calls for data exchange and flow control are processed.
-e	The CMX calls for handling events are processed.
-t	In addition to logging the error messages, the t_error() calls are processed explicitly. Error messages are always logged even if this option is not specified.
-v	The CMX calls, their arguments, the options and the user data are processed in detail. The extent of processing depends on the options specified for CMXTRACE. The calls and their arguments are processed without options and user data.
-x	The calls and their arguments are processed without options and user data.
-D	This option selects detailed processing with additional information about system calls.
file	Name of one or more files containing trace entries to be processed.

11.3 Example of Activating and Evaluating the CMX Library Trace

Example of a configuration for activating the CMX library trace:

```
csh: setenv CMXTRACE "-SD -p 0 -r 64 -f ."
```

or:

```
sh: CMXTRACE="-SD -p 0 -r 64 -f ."  
export CMXTRACE
```

Example of a configuration for editing the trace files:

```
cmxl -Dv CMX<pid> > file_name
```

It is advisable to redirect the data to a file, otherwise they are output to *stdout*.

➤ CMX error codes and a brief description can be found in the CMX header file */usr/include/cmx.h*.

12 Appendix

12.1 Complete list of CMX error messages

The following tables contain all possible CMX error messages, i.e. all error messages generated at the ICMX(L) program interface. The error messages are sorted by error type and error class.

Error messages with error class = T_DSSYSERR (5) or X_BX2 (8) are not defined in the header files of the CMX program interfaces. They can only be decoded using <errno.h>.

Error messages generated at ICMX(L)

Errors with error type (0) = T_CMXTYPE and error class = T_CMXCLASS (0):

Num. value	Symbolic value	Meaning
0	T_NOERROR	No error
4	T_ENOENT	All internally allocated resources are occupied.
5	T_EIO	The CCP is no longer operational.
12	T_ENOMEM	Not enough working memory.
14	T_EFAULT	Illegal address. One of the specified pointers does not point to the process address space.
22	T_EINVAL	Invalid argument.
100	T_UNSPECIFIED	Error not more precisely specified.
101	T_WSEQUENCE	Function call not permitted in this state.
102	T_WREQUEST	Illegal function call.
103	T_WPARAMETER	Invalid parameter.
104	T_WAPPLICATION	Unknown application or application already known under this name.
105	T_WAPP_LIMIT	No more processes in applications may be attached.
106	T_WCONN_LIMIT	Limit for connections reached.
107	T_WTREF	Invalid transport reference.
109	T_COLLISION	Collision in connection setup, disconnection, redirection or in sending data.
110	T_WPROC_LIMIT	Too many processes have attached applications.
111	T_NOCCP	No CCP present for desired application or connection.

112	T_ETIMEOUT	Waiting period elapsed.
113	T_WROUTINFO	Illegal CC list.
115	T_WRED_LIMIT	Too many simultaneously redirected connections.
116	T_WLIBVERSION	Version of CMX library used is incompatible.

Table 7: At ICMX generated error messages

In addition, in ICMX(L) the following errors, not belonging to error type = T_CMXTYPE and error class = T_CMXCLASS, may also occur:

Num. value	Symbolic value	Meaning
0	T_NOTSPEC	Error not more precisely specified.
1	T_DIRERR	The specified TS directory is unknown.
2	T_NAMERR	The specified name is not present or the specified name is already present.
3	T_ILLNAM	The name is syntactically invalid.
5	T_PROPER	The requested property is not present or the specified property is already present or syntactically invalid.
7	T_TIMEOUT	The TNSX daemon <i>tnsxd</i> does not respond within the time limit.
10	T_LEAFNO	More or fewer names were found than expected.
16	T_LENERR	One of the properties has an incorrect length.
20	T_PROT	Error in the protocol for <i>tnsxd</i> .
100	T_LFILE	The TS directory has an invalid format.

Table 8: Additional error values at ICMX(L)

12.2 List of reasons for disconnection

The reasons for disconnection passed by CMX in reason following the calls `t_disin()` are described below. The symbolic values specified here are numerically defined in `<cmx.h>`. The abbreviation CCP stands for "Communication Control Program", meaning the transport system. "Local CCP" stands for the CCP in the system of the current process, while "partner CCP" stands for the CCP in the system of the connection partner of the current process.

Reasons given by CMX

num. value	Symbolic value	Meaning
0	T_USER	Disconnection by the communication partner; possibly also due to a user error on the part of the communication partner
1	T_RTIMEOUT	Local disconnection by CMX due to inactivity on the connection as specified by the parameter t_timeout.
2	T_RADMIN	Local disconnection by CMX due to deactivation of the CCP by the administration.
3	T_RCCPEND	Local disconnection by CMX due to CCP breakdown

Table 9: Reasons for disconnection - given by CMX

Reasons given by the partner CCP

num. value	Symbolic value	Meaning
256	T_RUNKNOWN	Disconnection by the partner or the CCP; no reason specified.
257	T_RSAPCONGEST	Disconnection by the partner CCP due to a TSAP-specific bottleneck.
258	T_RSAPNOTATT	Disconnection by the partner CCP because the TSAP addressed is not attached there.
259	T_RUNSAP	Disconnection by the partner CCP because the TSAP addressed is not known there.
261	T_RPERMLOST	Disconnection by network administration or by adminof partner CCP.
262	T_RSYSERR	Error in network.
385	T_RCONGEST	Disconnection by the partner CCP due to resource bottleneck.
386	T_RCONNFAIL	Disconnection by the partner CCP due to failure in connection setup. Connection setup may fail e.g. because user data is too long or expedited data is not permitted.

387	T_RDUPREF	Disconnection by the partner CCP because a second connection reference was assigned for an NSAP pair (system error).
388	T_RMISREF	Disconnection by the partner CCP due to a connection reference that could not be assigned (system error).
389	T_RPROTERR	Disconnection by the partner CCP due to a protocol error (system error).
391	T_RREFOFLOW	Disconnection by the partner CCP due to connection reference overflow.
392	T_RNOCONN	Establishment of the network connection rejected by the partner CCP.
394	T_RINLNG	Disconnection by the partner CCP due to incorrect header or parameter length (system error).

Table 10: Reasons for disconnection - given by the partner CCP

Reasons given by the local CCP

num. value	Symbolic value	Meaning
448	T_RLCONGEST	Disconnection by the local CCP due to resource bottleneck.
449	T_RLNOQOS	Disconnection by the local CCP because quality of service can no longer be provided.
451	T_RILLPWD	Invalid (connection) password.
452	T_RNETACC	Network access refused
464	T_RLPROTERR	Disconnection by the local CCP due to a transport protocol error (system error).
465	T_RLINTIDU	Disconnection by the local CCP because it received an overly long interface data unit (TIDU) (system error).
466	T_RLNORMFLOW	Disconnection by the local CCP due to violation of the flow control rules for normal data (system error).
467	T_RLEXFLOW	Disconnection by the local CCP due to violation of the flow control rules for expedited data (system error).

